STRATEGIC DECISION MAKING UNDER UNCERTAINTY TAILORED TO PARALLAX CORRECTION AND ENERGY PRODUCTION

A dissertation submitted to

ETH ZURICH

for the degree of

Doctor of Sciences

presented by

BASTIAN HIERONYMUS MIGGE

Diplom-Informatiker Univ. (Technische Universität München)
born May 19th, 1980
citizen of Germany

accepted on the recommendation of:

Prof. Dr. Konrad Wegener, supervisor
Prof. Dr. Pavel Hora, co-supervisor
PD Dr. Andreas Kunz, co-supervisor

2013
"Damit das Mögliche entsteht, muss immer wieder das Unmögliche versucht werden."

Hermann Hesse, Steppenwolf

"Toen de èzel zich aan ut hóngerlieje gewèndj haaj, is der gestòrve."

Limburg’sches Sprichwort (Niederlande)
Acknowledgment

This PhD thesis would not have been possible without the help from many people. It was mainly funded by ETH Zurich and the research project 'W2E - commercial optimizer' and for multiple smaller contributions to various CTI and EU projects at our institute. I would like to thank the Institute of Machines Tools and Manufacturing and the CTI for funding my project.

First, I want to thank my family, my girlfriend and my close friends for supporting me unconditionally. Without you, this work would have never been possible. Thank you for taking care of me!

I thank my advisors Prof. Dr. Konrad Wegener, head of the Institute of Machines Tools and Manufacturing, Prof. Dr. Pavel Hora, head of the Institute of Virtual Manufacturing, and PD Dr. Andreas Kunz, head of the research group Innovation Center Virtual Reality for giving me the opportunity to work in research, science and teaching. I’m thankful for the confidence to let me try applying Artificial Intelligence methods in the fields of Human Computer Interaction and production systems.

I would like to thank my colleagues: Adrian Steinemann for research debates at the BMW IT Research Office in the USA and introducing me to the ETH Zürich. You motivated me to start searching.

Tim Schmidt for getting me into artificial intelligence in particular POMDPs and the time at PARC.

My roommate Thomas Nescher for the discussions about research in general, continuous finger pointing on problems and unanswered questions in detail as well as accepting personal limits and learn to focus.

It was a pleasure to share the office with you! You are right. 'Life is no Schoggi'.

Thomas Kennel for designing and building the interactive system digiBench, helping me to teach, for a never running dry and creative mind and for playing the bass.

Karl Ruhm for the most experienced senior researcher’s point of view. Thank you for listening and helping qualify my proposals. You motivated me to continue.

Georg Heinecke for discussions about production planning, the philosophy of research at jazz nights in Zurich.

Many students contributed to my work. I want to thank Oliver Stollmann for implementing AI software and bridging the gap between Python and MATLAB, and Philippe Zimmermann for helping me setting up a realistic model of the waste to energy plant KVA Turgi and evaluating the automatically learned strategies. I also thank Bruno Reinhart as well as Santiago Droll for conducting user studies at the interactive screens in order to deduce POMDP model from measurements of human behavior.

Furthermore, I want to thank Dr. Olga Pardo from ETH Human Resources for project management skills and constructive discussions beyond the hard facts of rationality. I also thank my project partners from ABB Switzerland Silvia Bardi, Esther Gelle and Christian Baltensperger for accompanying me.

Thanks to all of you for every minute of discussions, arguments and fun. I enjoyed writing this thesis; I enjoyed learning.
## Contents

Abstract xiv  
Zusammenfassung xviii  
Symbols xxi  
Abbreviations xxii  

### 1 Introduction 1  
1.1 Intelligent agents 2  
1.2 Non-myopic sequential decision making under uncertainty 3  
1.3 Outline 4  

### 2 Automated strategic decision making under uncertainty 7  
2.1 Probabilistic models of dynamic systems 8  
2.2 Simulation 9  
2.3 Optimal model predictive control and planning 10  
2.4 Markov decision processes 12  
2.4.1 Formal model definition 13  
2.4.2 Control policy 14  
2.4.3 Planning algorithms 15  
2.5 Partially observable Markov decision processeses 17  
2.5.1 Bayesian estimator 18  
2.5.2 Formal model definition 21  
2.5.3 Optimal control policy 22  
2.5.4 Automated planning 25  
2.5.5 Control under uncertainty 26  
2.5.6 Approximations for large systems 26  
2.6 Reinforcement learning 28  
2.7 Summary 29  

### 3 Related work 31  
3.1 Applications 32  
3.1.1 Industrial applications 32  
3.1.2 Autonomous systems 32  
3.1.3 Computer vision 33  
3.1.4 Assistant systems 33
3.2 Solving POMDPs .................................................. 33
   3.2.1 Solution methods ........................................ 34
   3.2.2 Solver software and toolkits ............................. 34
3.3 Relaxations .................................................. 35
   3.3.1 Belief space simplification ............................... 35
   3.3.2 Online planning .......................................... 35
3.4 Research gap ................................................ 36

4 Controller design ................................................... 39
4.1 Problem characteristics ....................................... 39
   4.1.1 Aspects ................................................ 39
   4.1.2 Simplifications .......................................... 41
   4.1.3 Automated planning under uncertainty .................. 42
4.2 Modeling the problem ......................................... 42
   4.2.1 Manual modeling ....................................... 44
   4.2.2 Automated model deduction .............................. 44
4.3 Multi-objective estimation .................................... 46
4.4 Modeling oblivion ............................................. 47
4.5 Summary .................................................... 52

5 Case study: Parallax error correction on interactive screens ........................................... 53
5.1 Artificial intelligence in human-computer interaction ................................................. 54
5.2 Human-computer interaction on interactive screens .................................................. 55
   5.2.1 Interaction space ....................................... 56
   5.2.2 Widget-based graphical user interface .................. 58
   5.2.3 Input hardware ......................................... 59
5.3 Problem definition ............................................. 60
   5.3.1 Parallax effect measures ................................ 64
   5.3.2 Potential improvement of interaction precision .......... 66
   5.3.3 Correction bounds ..................................... 67
   5.3.4 Automatically correcting the parallax error .......... 68
5.4 Related work .................................................. 68
   5.4.1 High precision direct touch interaction ................ 69
   5.4.2 Static display calibration .............................. 69
   5.4.3 Virtual pointer ......................................... 71
   5.4.4 Accurate interaction by hardware offset reduction .... 72
   5.4.5 Linguistic target reduction ............................. 72
   5.4.6 Avoiding the need for precise interaction .............. 73
   5.4.7 Irritation from spacial displaced haptic feedback ....... 74
   5.4.8 Evaluating the interaction quality of interactive systems .... 74
   5.4.9 Extending the interaction space ....................... 75
   5.4.10 Research gap .......................................... 76
5.5 Human user behavior identification .............................................. 77
   5.5.1 User viewpoint location ................................ 78
   5.5.2 User viewpoint dynamics ................................ 82
   5.5.3 Correlation between interaction and viewpoint ......... 84
   5.5.4 Interaction precision .................................... 87
   5.5.5 Focal point of area targets ............................ 89
   5.5.6 Conclusion ............................................ 93
Automated sequential decision making under uncertainty, such as model-based Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs), is an integral part of the research field artificial intelligence. It is used to find strategic control policies for planning and scheduling problems that consider uncertainty in terms of measurements of the control system’s actual state and the uncertain effect of an action onto the controlled system based on a scalar multi-objective optimization function.

POMDPs cover complex problems. However, solving them, more precisely automatically finding a strategic policy, causes high computational effort. To justify applying this method, the problem is required to exhibit complex characteristics, which are summarized in this thesis. In order to make POMDPs applicable to large models of real world problems the computational methods for automated planning under uncertainty are continuously improved. However, a-prior minimizing the underlying models makes the need for expensive calculations obsolete. But manually formulating probabilistic process and measurement models as well as cost functions is error-prone and not intuitive.

This thesis contributes four findings to applying sequential decision making under uncertainty to real world problems by focusing on the modeling process. They are theoretically developed and applied to case studies.

The first contribution is the working principle of applying a controller to an estimation problem. Controlling an estimate through planning under uncertainty enhances common estimation methods. They gain the ability to consider multiple, strategic goals, in adapting the estimate, which are represented as the controller’s optimality function. It is the cost for step-wise changing the estimate versus the resulting estimation error.

The second contribution is the development of an oblivion process for Bayesian filters. Due to the process model epoch length, model-based, time-discrete closed-loop control and estimation requires gathering observations, which are usually read out (pulled) from sensors. Event-driven (pushed) observations, however, do not necessarily provide a measurement at each control-cycle step. To deal with events a universal oblivion process for Bayesian filters is presented. If no observation occurred it lets the belief state converge stepwise against the uniform belief state, which represents that the controller has no knowledge about the current system state.

In order to apply POMDPs to realistic problems, the third contribution presents formal guidelines for the modeling processes for POMDPs. It is either manual modeling, passive model deduction from measurement data or active requesting an existing simulation. Once a model is given, automated planning is used to deduce a strategic control policy. However, it will be shown in this thesis that this approach is limited to problems, for which no simulation model could be generated, but measurements could be acquired (such as the human being in the first use case). In case of simulation models for complex real-world
problems, another approach for deducing control policies exists: Model-free learning skips the model
deduction step and directly deduces a control policy from actively challenging a real or simulated sys-
tem. This allows applying strategic decision making under uncertainty on the basis of complex models,
which can be easily formulated by process experts using intuitive tools such as MATLAB/Simulink®.

Deducing POMDP models from measurement data is applied to the field of human computer interac-
tion, in particular correcting the parallax error on interactive screens. The error stems from the user’s
viewpoint in front of the screen and affects the interaction precision on touch sensitive screens, due to an
offset between interaction plane and display plane. It is found that interaction errors indicate the user’s
viewpoint with uncertainty. Hence, the parallax error can be corrected by estimating the user’s viewpoint
through interaction errors instead of using additional, complex tracking hardware. According to the
previously introduced principle of controlling an estimate it can be formulated as POMDP by adapting
the estimation step-wise. The cost function covers the resulting parallax error and the user’s irritation,
which stems from changing the correction. Studying the user’s behavior in front of interactive screens,
a measurement based decision-theoretic model for reducing the parallax error on interactive screens is
presented.

The model is deduced from measurement data of studying the human behavior in front of a large interac-
tive screen. The studies investigated the user’s viewpoint position and dynamics, the correlation between
interaction error and viewpoint, the interaction precision for different touch techniques as well as the
focal point, which the user aims for, on area widgets (targets).

It is found that the user mainly moves in horizontal dimension with a distance of 57 cm to the screen.
The vertical position of the standing user is stable.

The analysis of the interaction position on the screen and the viewpoint in front of the screen shows that
right (left) handed people interact mostly right (left) of the viewpoint. It is proven by measurement that
the horizontal interaction error on the screen significantly correlates with the viewpoint position in front
of the screen.

To deduce the interaction error from elements of graphical user interfaces, which are commonly shown
as areas, they must provide a single focal point. For label buttons it is found that the focal point is lo-
cated at the geometric center of the widget area, independently of the label alignment. In contrast, the
interaction position on checkboxes is mainly distributed around the box on the left instead of the label.
It is, moreover, shown that the interaction error increases linearly with the size of the target.

Studying the user finally shows that the interaction error does not significantly differ between using a
pen tangible user interface and the bare finger for targets with the size of the human finger tip.

The fourth contribution is applying strategic control to complex, large problems. Since large problems
can hardly be manually expressed as POMDP models, model-free learners are connected to existing
simulations that use a high level modeling language. Integrating a high level modeling language enables
process experts to precisely express realistic problems. And learning allows deducing strategic control
policies with respect to uncertainty from systems that are unknown to the controller, in this case from
simulations. This approach enables to apply artificial intelligence to large, complex systems.

To evaluate this methodology, the following case study addresses market driven energy production of
a waste-to-energy plant, which is a complex dynamic system. The liberalization of the energy market
requires changing the production load quickly in order to stabilize the electricity grid. Large step load
changes, however, cannot be covered by common, yet proven control methods that are parameterized
around a well-studied operation point. In contrast, model based methods enable predictive control. The
system dynamics of the thermodynamic plant components and the unsteady heating value of waste as
fuel justify the use of stochastic control frameworks, like MDPs and POMDPs, to model and optimize
energy production strategically. As an example, a real plant is modeled and parameterized according
to measurement values in the high-level simulation software. This allows process experts expressing
the specific characteristics of the power plant. Then, a learner interacts with the simulation deducing a control policy for cost optimal energy production. According to the software-in-the loop principle, the deduced policies are loaded into a controlled and evaluated on the simulation.

Overall, the four findings of this thesis contribute to applying extended control methods from artificial intelligence to real-world problems. The novel methods are exemplarily applied in the fields of human computer interaction and industrial control. It is shown in practice how to deduce a POMDP from measurement data and deduce a strategic control policy applying automated planning on the POMDP model. It is, moreover, shown in practice how to deduce and evaluate a strategic control policy with model-free learning from a simulation. And the POMDP framework is theoretically extended in two aspects to solve estimation problems with respect to multiple objectives and to deal with event observations that occur irregularly.
Zusammenfassung

Maschinelle sequentielle Entscheidungsfindung unter Unsicherheit, wie Markov Entscheidungsprozesse (MDPs) und Partiell observierbare Markov Entscheidungsprozesse (POMDPs), sind ein integraler Bestandteil des Forschungsbereichs Künstliche Intelligenz. Sie werden genutzt, um strategische Kontrollregeln für Planungs- und Terminierungsaufgaben hinsichtlich einer skalaren, mehrdimensionalen Optimierungsfunktion zu erhalten. Dabei werden sowohl Unsicherheit in Form von ungenauen Messungen des aktuellen Systemzustands als auch unsichere Effekte von Aktionen auf das zu steuernde System berücksichtigt.


Der erste Beitrag ist die Anwendung eines Reglers auf ein Schätzproblem. Einen Schätzer durch Planung unter Unsicherheit zu kontrollieren erweitert die heutigen Methoden um die Einbeziehung mehrerer, strategischer Ziele, die als Optimierungsfunktion des Reglers formuliert sind. Die Optimierungsfunktion wägt die Kosten für die schrittweise Anpassung der Schätzung durch Aktionen des Reglers gegenüber dem resultierenden Fehler der Schätzung ab.


Um POMDPs auf realistische Probleme anzuwenden, präsentiert der dritte Beitrag einen Leitfaden für den Modellierungsprozess. Entweder werden POMDPs manuell modelliert, passiv von Messdaten oder aktiv von Simulationen abgeleitet. Aus dem Modell wird dann mittels Planung eine strategische Kon-
Zusammenfassung


Symbols

\[ [i, j] \quad \text{Interval between } i \text{ and } j \text{ over the space of } i \text{ and } j, \text{ including } i \text{ and excluding } j \]
\[ |x| \quad \text{magnitude of } x \text{ or number of element of a set} \]
\[ 1 \quad \text{Unit matrix or one vector in adequate dimension} \]
\[ A \quad \text{Action space} \]
\[ a \quad \text{Action} \]
\[ \alpha \quad \text{Smoothing factor of exponential smoother} \]
\[ \arg \max \quad \text{Argument of the maximum} \]
\[ B \quad \text{Belief space (or Information space)} \]
\[ b \quad \text{belief state (or belief), POMDP state} \]
\[ \epsilon \quad \text{Balance factor between exploration and exploitation of learning policy} \]
\[ \gamma \quad \text{Discount factor for planning and learning} \]
\[ I_{x,y} \quad \text{Interaction position in display dimensions } x,y \]
\[ J_{x,y} \quad \text{Interaction position in display dimensions } x,y \text{ relative to the target position} \]
\[ LI \quad \text{Load indicator} \]
\[ \log \quad \text{Logarithm} \]
\[ O \quad \text{Observation model (measurement model)} \]
\[ o \quad \text{observation} \]
\[ O() \quad \text{O-Notation defines the complexity class of an algorithm.} \]
\[ P() \quad \text{Probability mass function} \]
\[ P \quad \text{Electrical power} \]
\[ PI \quad \text{Power indicator} \]
\[ pI \quad \text{Pressure indicator} \]
\[ p \quad \text{Pressure} \]
\[ \dot{Q} \quad \text{Heating power} \]
\[ R \quad \text{Reward model} \]
\[ \mathbb{R}^n \quad \text{n-dimensional space of real numbers} \]
\[ S \quad \text{State space} \]
\[ s \quad \text{MDP state} \]
\[ \sigma \quad \text{Standard deviation, if not state otherwise} \]
\[ T \quad \text{Transition model (system dynamics)} \]
\[ T \quad \text{Temperature} \]
\[ T \quad \text{Temporal horizon} \]
\[ TI \quad \text{Temperature indicator} \]
\[ T_{x,y} \quad \text{Focal point of target in display dimensions } x,y \]
\[ t \quad \text{Time} \]
\[ \Theta \quad \text{Observation space} \]
\[ V_{x,y,z} \quad \text{User viewpoint in } x,y,z \text{ dimension relative to the target position} \]
Abbreviations

AI  Artificial intelligence
dpi  dots per inch
GUI  Graphical user interface
h  Hour
HCI  Human-computer interaction
HMM  Hidden Markov model
Hz  Hertz
K  Kelvin
kg  Kilogram
Kinect  Microsoft®Kinect®sensor
KVA  Kehrichtverbrennungsanlage (waste to energy plant)
LCD  Liquid crystal display
m  Meter
MC  Markov chain
MDP  Markov decision process
MJ  Megajoule
mm  Millimeter
MW  Megawatt
POMDP  Partially observable Markov decision process
px  Pixel
TUI  Tangible user interface
w.l.o.g.  without loss of generality
WtE  Waste to Energy
w.r.t.  with respect to
ppi  Pixel per inch
px  Pixel - basic unit on digital displays
s  Seconds
Introduction

Artificial Intelligence (AI) aims to create intelligence with technological systems called agents; focusing on the agent’s internals: problem solving by searching, knowledge representation, logic reasoning and planning (under uncertainty), learning and communication [184].

Today, designing agents to accomplish complicated tasks is one of the major goals of the research fields of AI in Computer Science and Control Systems in Mechanical and Electrical Engineering.

To realize a goal, the agent takes input signals from the environment and applies output actions to it. A policy represents the agent’s decision strategy of how to act. It defines the outputs with respect to the given inputs. The basic principle of agent interaction is in closed loop with the environment: getting an observation and selecting the best action for execution.

For instance, an automated window shutter system controls the position of a window shutter to reduce the sunshine intensity inside a house using a brightness sensor. A simple reactive strategy is to close the shutter if the brightness is measured above a threshold and open it otherwise. This simple strategy might annoy the people in the house: On a partly cloudy day the brightness may oscillate around the threshold; the controller will continuously move the shutter that might make annoying noise. The simple controller did not take into account that closing the shutter before immediately open it again does not make sense at all. This example shows that even a simple task needs to take into account the relationship between present decision and future consequences to achieve a good overall performance. Developing a suitable strategic control strategy for complicated systems is far from obvious.

The input values are provided by sensors, which give inaccurate and incomplete measures due to physical limitations and not evaluating relevant environmental aspects. In sequential decision making, the agent recursively estimates the real environment state from observations and applies an action with respect to a control law in a control loop. Like observations, the effect of the actuator’s actions may be uncertain, too.

Deducing a strategic control law is a fundamental aspect of designing control systems. Non-myopic controllers do not only consider the immediate but the long-term effects of an action by planning out the future system behavior based on a model. The model describes the (uncertain) system dynamics under the effect of actions, uncertain observations and a cost function which takes into account multiple goals for a trade-off between (conflicting) objectives. In particular, Partially Observable Markov Decision Processes (POMDPs) is a framework that covers sequential decision making under uncertainty.
1 Introduction

1.1 Intelligent agents

The word *agent* stems from the Latin word *agere* (to do, drive, lead, pass or spend time). ‘The ability to make predictions about the future [...] is the crux of intelligence.’ [114]. So intelligent agents are things that act with respect to future effects. In computer science, agent systems are a piece of software and hardware which interact with their separately defined environment. The interaction – executed by hardware – is divided into observations and actions: Actions are triggered by the agent and effect the environment; observations inform the agent about the status of the environment as a reaction from the agent’s point of view. Trying to influence the environment to perform optimally with respect to previous defined performance criteria [184] [171] is, on the other hand, done by software.

The behavior of the agent system stems from rules that define decisions about how to act in certain situations to fulfill a certain goal. For instance, a robot tries to reach a certain location, or the climate control tries to reach a certain room temperature. This is formally represented by a policy that defines the action to be done in a given situation. The policy represents the intelligence of the agent.

The concepts and methods of how to design and implement intelligent agent systems are developed in the research field of Artificial Intelligence (AI) as part of Computer Science. Driven by the two fundamental aspects of informatics: Data representation and data processing, AI creates intelligent machines by developing:

- high quality (adaptive, fast, scalable, robust etc.) algorithms to process information and
- effective (flexible, scalable etc.) representations for large (and discrete) data sets of detailed information.

Sorting and searching with computers are fundamental, well studied aspects in computer science [121]. By searching in models that represent the environment, the agent is exploring the behavior of the world and uses its structure to ‘find sequences of actions that achieve its goals, when no single action will do’ [184]. Exploring the future effects of actions on the environment is called planning. It is methodologically strongly related to shortest path problems in graph theory. It enables to automatically deduce a control policy from the system dynamics. Motivated by the fact that the agent’s sensors do not give perfect information, the agent can only assume its situation. Considering that uncertainty into the development of a strategic policy is called planning under uncertainty. Assuming that the knowledge base about the environment of the agent is not perfect, a learner adapts its incomplete knowledge to the actual environment. Hence, learners develop the policy from challenging the controlled system. And approximate planner, such as online planner, reduced the planning effort by investigating only the current situation. Based on these techniques, today, multi-agent systems with distributed intelligence, sensors and actors are in the focus of AI research.

Historically, Strong AI tries to create universal intelligent systems inspired by humans. A first step was to emulate neurological findings on information systems (Cybernetics). Reducing the computational efforts, the development of symbolic methods assumes that intelligence can be expressed by manipulating symbols. Besides the development of logic inferring systems in programming languages (like Prolog), knowledge based systems (e.g. expert systems) lead to the first successful applications of AI [184]. Since the systems use knowledge based on human expertise to solve problems, this approach is - by definition - limited to specific problem domains. On the other hand, general approaches lead to highly large and complex systems. Although the computational capacity steadily increases (Moore’s law), the required computations still take too long in practice. To reduce that, problem specific information is actually used to develop approximation (heuristic) methods trading off accuracy for calculation effort of complex problems [98].
1.2 Non-myopic sequential decision making under uncertainty

Today, many research areas work on intelligent control systems; each one with a specific focus. AI focuses on the effective implementation of intelligence on computing machines in terms of algorithms and storage in a generic, numeric way. In contrast, control systems – as part of mechanical and electrical engineering – focus on modeling physical aspects of the environment (called system) as continuous system dynamics (linear and non-linear) and estimators to segment noisy sensor values. Closed-loop state trackers (e.g. Kalman Filter, Particle Filter) are interpreted as signal filters that (i.e. recursively) map noisy sensor measurements to well defined system states using statistical estimators, e.g. maximum likelihood estimator or the Kalman filter.

In this context, statistical classifiers, developed in the field of Machine Learning, are the trainable successors of signal filters based on more complex underlying mathematical methods [77]. The algorithmic aspect of intelligent systems - planning the control policy - is handled pragmatically: It is separated and simplified using common techniques (e.g. Linear Quadratic Regulators) on linearized models, which is proven to be optimal by the separation principle under certain circumstances.

A widely used approach in industry are controllers without knowledge about the system dynamics (e.g. proportional-integral-derivative (PID) controllers). Such controllers work well on continuous systems but do not take into account strategic effects of the control, which is reasonable controlling physical systems, e.g. a valve, around a certain operation point. Besides acting myopic, the drawback of such controllers is the unintuitive parameter tuning, which is mostly done automatically since manual tuning methods can be relatively inefficient. As a result, it prevents further development of the controller (’Never touch a running system.’).

In contrast to AI, engineering disciplines focus on investigating and modeling the environment, sensors and actuators of control systems instead of developing data structures to store them or algorithms to calculate policies. As indicated in the different language for intelligent systems in Table 1.1, the most important difference between AI and Control Systems is the point of view. Since computer scientists develop the internals of intelligent agents, the point of view is from the inside of the agent to the outside; they describe the environment as external. On the other hand, the engineering disciplines view is the other way around: They need to apply some effective (intelligent) instrument to control their system (environment).

1.2 Non-myopic sequential decision making under uncertainty

Sequential decision making is a process (of a potentially infinite amount of time steps) in which an agent at each step receives some information about the world and selects an action based on the accumulation of this information. The agent considers the temporal aspect of the controlled system. A discrete agent has to choose from a finite set of actions on how to interact with the world. In general, the information received is incomplete (hence, does not allow deducting directly the actual state of the world) and the results of interactions can be uncertain.

An optimal policy for such a process is a mapping of the accumulated information to action choices that maximizes the expected value of some valuation-function. Besides this process oriented optimization, task oriented problems focus on reaching a certain goal, which is expressed as state with a high terminal reward.

A common formalization for such problems is a Partially Observable Markov Decision Process [196]. Markov Decision Processes (MDPs) have proven to be useful in a variety of sequential planning applications in which it is crucial to account for uncertainty in the process [171]. Partially Observable MDPs generalize this paradigm with a sensor model that expresses the uncertainty of sensing the environment’s
real state (see Chapter 2).

The motivation for using uncertainty – expressed as random variables (with a certain distribution, e.g. Gaussian, Poisson, Brownian, Erlang) – in modeling the world is that the nature of stochastic has two possible reasons. Either the world’s nature actually is stochastic (stochastic process), or the understanding of the system is limited and random variables to compensate the uncertainty in modeling (expressing informations) the world are used.

Based on a model, a planner searches for the optimal control law by simulating the process the model describes. The benefit of probabilistic planning is that it does consider uncertainty in developing the control strategy. That means although an action might lead to a state with high reward (e.g. a lottery win), it does take into account the probability (in this case - the low probability) of eventually reaching the state. Hence, the controller might not spend money (immediate costs) to buy a lottery ticket but instead invest in research projects, that might lead to a rewarding industry product. A detailed introduction in the method of planning under uncertainty with POMDPs is given in Section 2.

Applications that utilize strategic (intelligent) controllers are for exampled automated scheduling of production processes, game planning (backgammon, chess) or autonomous vehicles (Mars rover, Google driverless car) and many more (see Chapter 3.1). So far, the research focused successfully on domain specific solutions instead of general problem solving like machine self-education [94] proposed as strong AI.

1.3 Outline

Markov Decision Processes take into account the uncertain affect of actions onto the controlled system. And POMDPs add the nature of uncertain measures of the current system states combining the concept
1.3 Outline


The benefit of applying POMDPs is that a planner automatically deduces a non-myopic/strategic control policy from a model. The model defines the controlled system which expresses the structure of the problem and immediate effects (dynamics and costs), such that the modeling engineer does not need to understand their long-term effects. POMDP models are either defined manually or deduced automatically from other models or measurement data (system identification). Reinforcement learning, however, combines policy deduction and system identification.

In this thesis advanced control techniques are discussed, theoretically extended and applied to two widely different use-cases: Parallax error correction in the field of human-computer interaction and demand driven energy production in the field of industrial control.

Chapter 2 introduces the method of POMDPs. Starting from probabilistic models for expressing system dynamics, model predictive control and Bayesian estimators, MDPs are combined with hidden Markov models to, finally, define POMDPs.

Chapter 3 gives an overview on MDP and POMDP applications. The common solution methods show the development of POMDP algorithms. Reducing the high computational effort of solving POMDP is one approach to make the method applicable to industry problems. In particular, relaxations in terms of modeling and planning are introduced. The chapter concludes with upcoming research questions.

Chapter 4 focuses on modeling POMDP. It shows the problem characteristics that require POMDP, and defines the process of modeling POMDP based on deducing the dependencies of the model elements.

In Section 4.1, the required problem characteristics, that make a POMDP necessary, are listed and simplifications are discussed. This answers the question, when to apply POMDPs. How to methodically bridge the gap between (large) real world problems and effective (small) models is answered in Section 4.2. It describes the dependence of the model components and discusses the modeling process, in terms of manual modeling, which is the most common technique, automated model deduction and automated policy learning, which allows to express the problem in a high-level modeling language instead of a unintuitive POMDP model.

Theoretical extension for POMDP controllers are discussed in Section 4.3 and 4.4. Section 4.3 introduces the working principle of estimation controller, which extend common estimators (filters) by a multi-objective cost function that effects changing the estimate. Section 4.4 introduces a oblivion process for event based observations. Commonly, controller gather sensor information at each control loop step to gather the system state and apply optimal control appropriately. Event-based observations, however, do not provide information at each step. In this case an oblivion step is applied instead of an observation. For a Bayesian filter the oblivion process is developed as a linear function and proven to converge against the uniform belief state, which represents no knowledge about the system state. The convergence speed of the oblivion function is, moreover, proven to be adjustable by a scalar parameter of the linear transition function.

In Chapter 5, a POMDP is practically applied to the problem of calibrating interactive surfaces in the domain of human-computer interaction. It corrects the parallax distortion on interactive screens controlling the estimate of the user’s viewpoint, from which the parallax error correction is deduced depending on the interaction position on the screen. The controller adapts the viewpoint estimate according to observed interaction errors on the screen. The underlying model is defined manually regarding the topology (state space, observation space, rewards) and it is deduced from measurement data of user studies regarding the transition and the observation model in Section 5.5. In this chapter is practically shown, how to deduce POMDP models from measurement data.
Since, changing the estimate reduces the parallax error on the one hand, but changing the correction irritates the user on the other hand, the given problem has multiple objectives, which common estimation methods do not cover. Instead, a POMDP controller is applied according to the theory introduced in Section 4.3.

Correcting the parallax error based on interaction errors, is no typical use case for recursive estimation (and control), since the observations can not be pulled from a sensor by the estimator at each step in the loop. It might happen, that the user did not interact within the update rate of the estimator. To deal with such event driven observation system, an oblivion model is introduced in Section 4.4. If not observation occurred, the model develops the belief state of a Bayesian Filter step-wise towards back to the uniform belief state, which represents no knowledge about the real system state. This effects the control action to be more restrained.

In Section 6, it is practically shown, how to automatically deduce a strategic control for large, real world systems, such as economically optimal energy production of a combustion plant. The complexity of large problems can, however, only be modeled in high level modeling languages. In order to bridge the gap between high-level modeling languages and complex, strategic controllers, in particular MDPs and POMDPs, a software framework is developed. It integrates the simulation in MATLAB/Simulink® and controllers in Python, such as planning respectively learning algorithms. This enables to automatically deduce and evaluate strategic control policies under uncertainty from complex, realistic models in MATLAB/Simulink®.

To test the framework on a realistic problem, a model of the waste-to-energy plant KVA Turgi is developed in MATLAB/Simulink®. It describes the system dynamics and the cost function of the system. The system is controlled under uncertainty, due to the uncertain heating value of the burned waste, which defines the overall energy production of the plant. Due to the system dynamics, which are caused by the thermodynamic behavior of the plant component, strategic control is required. Hence, the methodology of MDPs is needed. However, due to it’s complexity, the underlying model can not be manually expressed as Markov decision process. Hence, the control strategy is automatically deduced from a MATLAB/Simulink® simulation of the plant using model-free Reinforcement learning. The plant model is parametrized with measurement data from the real waste-to-energy plant KVA Turgi. Then, the model is simplified in order to reduce the computational complexity. And finally, the control laws from different learner setup are deduced and evaluated on the simulation.

The work closes with a summary and a conclusion of applying sequential decision making to human-computer interaction and production control.
Automated strategic decision making under uncertainty

This section describes the aspects of strategic sequential decision making under uncertainty, joining the idea of model based controllers with uncertainty in actions and observations with planning that enables non-myopic decision making. Partially observable Markov decision processes (POMDP) are a framework to express such problems.

Controllers are systems with the purpose to control a real world system, e.g. a power plant or the calibration of a human computer interface. This controller follows a control law selecting actions which are then executed by an actuator. To select the action properly, a closed loop controller is equipped with sensors which gather information about the system’s current state. A closed control loop is defined as follows: At a certain point in time (stage), the controller gathers the state of the environment. Based on this state, it selects and executes an action that a) affects the evolution of the environment and b) results in an immediate reward to the agent [171]. The reward is crucial for optimal control, as it will be introduced below. Figure 2.1 shows the process of controlled process. In contrast, an open loop control does not gather information about the current state of the controlled system. It applies the control actions based on a control law 'blindly' (without feedback).

The following example shows the benefits and challenges of closed-loop control. To control, for instance, the hallway light, an open-loop controller could switch on and off the light at certain daytime modeling the progress of time internally. Since twilight does not take place at the same time every day, this controller might not work satisfactorily. Adding feedback from the real environment in terms of a light sensor enables the controller to observe the system state (closed-loop controller). This makes the time (system) model obsolete, which is widely used in industrial applications by the family of PID controllers. PID controllers observe and error as difference between the measurement and the target set

![Figure 2.1: Sequential decision process](image)

State $s(t)$ Action $a(t)$ Reward $r(t)$ Action $a(t+1)$ Reward $r(t+1)$
point of a controlled variable of the environment. Instead of a system model, it uses the weighted accumulation of the current error, the integration of the error history and the future error value (by differential extrapolation) to deduce robust control. For instance, a robust rule is to not switching off the hallway light if a car light just glares the sensor.

In contrast to explicitly expressing the control law, optimal control addresses the task of automatically finding a control law for a given system model w.r.t. an optimality criterion. Modeling the system allows non-myopic control by simulation: Exploring the system dynamics enables to predict the system’s progress based on the current state (model predictive control). An automated planner deduces it from a given model describing the system dynamics under the effects of actions and a cost function. Considering uncertainty extends the controller to select actions carefully. Hence, the policy is not defined manually but automatically deduced from a model.

In this chapter, probabilistic system dynamics, simulation and model predictive control is introduced in principle before Markov decision processes under uncertainty are presented in detail.

### 2.1 Probabilistic models of dynamic systems

A system model represents a real world system. Usually the model is an abstraction, limitedly describing the relevant aspects of the real world system. A process model describes the system behavior over time - the system dynamics. It expresses the states of a system - either continuous or finite - and its dynamics as transition function from a state to its successor state. Controlled processes are modeled as transition function that does additionally depend on the control action.

Deterministic system models describe the system dynamics with deterministic transitions: A state is processed into a single successor state. In deterministic controlled processes, the outcome of actions is also certain. Hence, the transitions are labeled with the corresponding action.

Stochastic models extend the system dynamics to be uncertain, due to exogenous or not modeled effects onto the system behavior or the uncertainty of the action’s effect. A controlled boat at sea, for instance, does not necessarily move straightly forward as forced by the engine due to current. It is, however, reasonable to not model the current to steer the boat position for simplicity reasons. The model is an abstraction of the real world system. Nevertheless, the current can be covered as exogenous aspect adding uncertainty to the effect of actions - in this case the position of the boat - using a probabilistic model. It is, moreover, possible that the real world itself is a random process. However, in such models, the system state is expressed as random variable and transition functions model the system’s dynamics. The state of a stochastic model is mathematically expressed as random variable \( P(X) \). A stochastic process model, however, defines the system state w.r.t. the system dynamics. It is mathematically modeled as conditional probability \( P(X_t | \ldots) \). Usually the state at time \( t, P(X_t) \) depends on the predecessor states of the system in time.
Stochastic processes that fulfill the Markov property can be modeled as Markov chains. A stochastic process \( \{ X_t \}_{t \in \mathbb{N}} \) over the state space \( s \in S \) has the Markov property if the system state at time \( t + 1 \) does only depend on the system state at time \( t \):

\[
P(X_{t+1} | X_0 = s_i, X_1 = s_j, \ldots, X_t = s_k) = P(X_{t+1} | X_t = s_k)
\]

for every sequence \( s_i, s_j, \ldots, s_k \) [100]. A stochastic process that fulfills the Markov property is called a Markov chain.

The order of the Markov chain expresses the number states backward in time the current state depends on. Dependencies of more than one time step are modeled by extending the state space. One artificial state merges a set of instances of real states. Markov chains with order zero model the system states independently from the predecessor in time.

Time invariant Markov chains neglect the dependence on a specific time step and express the process independently of stage (time step \( t \)). The transition probability \( P(X_{t+1} | X_t = s_i) \) is equal for all \( t \).

For simplicity reasons, the probability distribution of the successor state of \( s_k \) at time \( t \) : \( P(X_{t+1} | X_t = s_k) \) is denoted as \( P(\cdot | s_k) \). Time invariant (or stationary) processes denote the transition probability for a given state \( s_k \) as \( P(\cdot | s_k) \) and the probability of the process taking state \( s_{sink} \) after state \( s_{source} \) within one time step is denoted as \( P(s_{sink} | s_{source}) \).

A Markov process over a finite state space \( S \) is modeled as probabilistic matrix \( T \in |S| \times |S| \). The elements \( s_{i,j} \) express the transition probability from (to) state \( s_i \) to (from) state \( s_j \) as conditional probability. Each row (column) sums up to 1. Hence, \( T \) is a stochastic matrix. The probabilistic character of the model allows considering uncertainty, as already discussed.

A model joins single aspects of a system, such as dynamics and states, that must be expressed manually by an expert. However, the system dynamics are expressed step-wise. Based on such a model the long-term behavior, can be evaluated by simulation without explicitly modeling it receptively understanding it.

### 2.2 Simulation

Due to the *The Stanford Encyclopedia of Philosophy* [219] simulation in science is defined as follows: 'In its narrowest sense, a computer simulation is a program that is run on a computer and that uses step-by-step methods to explore the approximate behavior of a mathematical model. Usually this is a model of a real-world system (although the system in question might be an imaginary or hypothetical one). Such a computer program is a computer simulation model. One run of the program on the computer is a computer simulation of the system. The algorithm takes as its input a specification of the system’s state (the value of all of its variables) at some time \( t \). It then calculates the system’s state at time \( t + 1 \). From the values characterizing that second state, it then calculates the system’s state at time \( t + 2 \), and so on. When run on a computer, the algorithm thus produces a numerical picture of the evolution of the system’s state, as it is conceptualized in the model. [...]’

Hence, simulation experiments enable to understand the behavior of the system or evaluate strategies for the operation of the system; in terms of applying actions onto the system. Starting the simulation with the current state of the real system allows predicting the future behavior of the system, which enables model predictive control.

As already introduced, controlled processes are system dynamics which are effected by a control action. A simulation of a controlled process returns the immediate reward and the next system state on an action.
Simulating a controlled process enables to automatically investigate the long-term effect of an action. A control policy expresses the law of a controller in which state which action should be applied. Developing a strategically optimal control policy from simulation is called planning, as it will be introduced in the next section.

In general, by simulating a system, system improvements are examined without the need of changing the real system. This reduces the risk of unexpected behavior and the costs of system modification.

It is important to note that a temporal simulation on probabilistic models only samples the modeled random variables (stochastic simulation). The outcome is one trajectory (chain of system states (and actions)) with a certain reward per run. To approximate the system behavior, multiple runs are done with changing initial states. The result is a random variable over the investigated system variables.

### 2.3 Optimal model predictive control and planning

Model predictive control (MPC) enables considering the future of the controlled system’s behavior in deducing the control policy. It deduces strategic control laws planning out (simulating) future costs from a model of the system dynamics. Although the methods of MPC are complex compared to a simple controller like PID controller, MPC controllers are capable to deal with complex system dynamics and large time delays.

Optimal control is the process of finding a control law for a given system based on mathematical optimization methods. Its general approach is minimizing (maximizing) a cost (reward) function over time, subject to the system dynamics. A common method to address this class of problems is a Linear quadratic regulator for deterministic as well as stochastic problems over a finite and infinite horizon. It models linear system dynamics and quadratic costs and solves the optimal control directly, using the differential Riccati equation. However, this approach is not applicable for a limited set of control actions.

The process of finding an optimal control law with respect to the future behavior of the system [184] under restricted actions is called planning. Hence, the resulting policy is commonly named strategy instead of law. Automated planning is the computational process of deducing a control strategy by searching for the optimal control strategy on a given model of the system behavior. The key idea of optimal control using planning is that the optimal control trajectory is a combination of optimal sub-trajectories (Bellman equation), which leads to the approach of dynamic programming in solving the planning problem automatically.

Figure 2.2 shows the key idea of planning for a discrete time system. The system state (ordinate) is changed by an action at each time step. To deduce the control law at time \( \text{now} \), the planner figures out which action (\( a1 \) or \( a2 \)) is better considering a cost function. In this example, the costs are defined as offset between the system state (shown as circle) and the target (shown as dotted line). Hence, a controller that minimizes the costs will try to bring the current state in line with the target state.

Figure 2.2(a) shows the immediate costs of both actions. In this case, action \( a1 \) is closer to the target. Hence, the control policy would express that \( a1 \) should be applied in the state \( s \) at time \( \text{now} \), since the planner only considers the immediate costs of the system state in the next time step \( \text{now}+1 \). Such a policy is called to be greedy. In contrast, a non-myopic policy is deduced from considering the summation of future costs over (in this case) a finite horizon. The upcoming costs are simulated on the system model, as shown in Figure 2.2(b). (For simplicity reasons, the future system dynamics are not considered, although this is usually not realistic.)

In the following, different types and aspects of planning are introduced.
2.3 Optimal model predictive control and planning

![Diagram](image_url)

(a) Immediate costs

(b) Accumulated costs

**Figure 2.2:** Accumulated cost function of model predictive control

In *classical planning*, the planner tries to reach a certain goal state from a given starting state over a finite horizon on a deterministic model. Navigating a robot to a certain position is such a problem. Since the goal state is absorbing, the planning process is stopped after reaching the goal state. The classical approach does not take into account the uncertainty of action effects and sensor information about the current state.

*Optimal deterministic planning* reformulates the planning problem from reaching a goal state, to rather optimizing the trajectory with respect to an additive reward function. The reward function models immediate rewards (or costs) for each system transition. Thus, planning problems can be solved with shortest path algorithms, such as $A^*$. Planning is expressed as optimality problem with one infinity rewarding state - the goal state.

*Conditional planning* does not assume the planner to know the initial state. It develops optimal plans for several states. Finite horizon conditional plans define the optimal \{state, action\}-trajectory with a length of the given horizon. In case of a finite state and action space, all conditional plans can be expressed
as directed graph (or state machine) each. To reduce the storage size, conditional plans are defined recursively by expressing the best action for a given state and referring to the conditional plan of the successor state.

Probabilistic planning takes into account the probabilistic behavior of the system in terms of uncertain effects of actions, exogenous events or the uncertainty of measuring the system state. In contrast to simulating the dynamics of a probabilistic system by sampling a state instance from a random variable and loosing the information of the randomness, probabilistic planning develops a probability distribution of the system state over time. Hence, the system behavior is captured completely.

The benefit of probabilistic planning to develop a non-myopic control policy is that the controller does take into account the uncertainty of the system dynamics, being implicitly expressed in the non-myopic control law. Moreover, non-myopic policies are not intuitive like the dynamics of a simulation. In contrast to formulating the control law manually, planning deduces optimal control from just the system model.

The drawback of naive planning is the high computational effort. Hence, optimizing planning algorithms is a well studied research field. It was stated that the benefit of automated planning is needed to only model the system dynamics. As shown, probabilistic models are expressed as large stochastic matrices that are not intuitive to handle. To simplify the models in terms of reducing the complexity of time dependence is reasonable for a large field of problems. Hence, the Markov property is widely applied on frameworks such as MDPs and POMDPs.

### 2.4 Markov decision processes

Markov Decision Processes (MDPs) - in the context of POMDP also called Fully Observable Markov Decision Processes (FOMDPs) - are defined as 'controlled stochastic processes satisfying the Markov property and assigning reward values to state transitions' [196]. MDPs are based on Markov chains.

Markov Chains are a widely used approach to model memoryless random processes of states and transition probabilities between states (see Chapter 2.1). The Markov property assures that a state only depends on its direct predecessor. Transitions model the probability of a time invariant system moving from on state to another state \( s_{source} \rightarrow s_{sink} \) within one time step as conditional probability distribution \( P(s_{sink}|s_{source}) \) for time-invariant systems.

To describe a sequentially controlled, stochastic dynamic system, the framework of Markov Decision Processes (MDPs) extends Markov Chains by modeling the affect of actions onto the system and a cost function [171][52]. The affect of the control input \( a \) onto the behavior of the system is expressed as conditional probability distribution that does not only depend on the predecessor state \( s_{source} \) but additionally on the control input \( a \). Hence, the process is modeled as \( P(s_{sink}|s_{source}, a) \): the state transition function is modeled for every action of the controller w.r.t. uncertainty of the action’s effect. A cost (or reward) function defines the immediate costs of a transition. The cost function expresses the value of the system behavior deduced from the controller’s goals.

Based on such a model, a planner automatically deduces an optimal policy for controlling the system, as discussed in the following section.
2.4 Markov decision processes

2.4.1 Formal model definition

MDP models describe the controlled system dynamics and the immediate reward of an action. Being used from and agent to choose actions at discrete time steps (stages), the system is modeled discrete. Formally, a MDP is defined as 4-tuple \((S, A, T, R)\):

- \(S\) is the state space of the dynamic system
- \(A\) is the space of actions applicable to the system
- A set of state transition probabilities given by the function \(T : S \times A \times S \rightarrow [0, 1]\)
- A reward function \(R : S \times A \times S \rightarrow \mathbb{R}\)

The state and action space are assumed to be finite.

To model a controlled system dynamics, the effect of each action is modeled separately adding the action space to the transition function \(T\). Hence, the transitions model the system dynamics under the effect of actions and exogenous effects which are beyond the control of the agent. For simplicity reasons, the set of applicable actions in each stage is assumed to be the same - the full set of actions of the agent. \(T\) expresses the probability of reaching the sink state for applying an action to a given source state:

\[
T(a) = \begin{pmatrix}
P(s_0|s_0, a) & \cdots & P(s_0|s_{-1}, a) \\
P(s_1|s_0, a) & \cdots & P(s_1|s_{-1}, a) \\
\vdots & \ddots & \vdots \\
P(s_{|S|-1}|s_0, a) & \cdots & P(s_{|S|-1}|s_{|S|-1}, a)
\end{pmatrix}
\]  

(2.2)

As for Markov processes, the probability distribution over all successor states \(s_{sink}\) follows the fundamental property that gives the Markov decision process its name: Writing the history of states and action from time step 0 until time step \(t\) as \(h_t = (s_0, a_0, \ldots, s_{t-1}, a_{t-1}, s_t)\), then the probability of reaching \(s_{t+1}\) with respect to action \(a_t\) depends only on \(s_t\) and \(a_t\), and not the entire history \(h_t\). If the conditional probability of the occurrence of event \(x\) given that \(y\) is true is written as \(p(x|y)\), then the Markov property holds:

\[
\forall h_t, a_t, s_{t+1} : P(s_{t+1}|h_t, a_t) = P(s_{t+1}|s_0, a_0, \ldots, s_t, a_t) = P(s_{t+1}|s_t, a_t)
\]  

(2.3)

As illustrated in Figure 2.3, MDPs can be represented by a finite set of Markov chains. Each Markov chain symbolically represents a state machine. State machines consist of a set of states - represented as nodes (graph theory) - that are pairwise connected through transitions - represented as directed edges. The states represent the status of the modeled system at a certain time. The transitions represent the system dynamics over time: It models the system going from one to another state within a time step with a certain probability. The probability expresses the non-determinism or stochastic character of the system dynamics. Hence, the probabilities of all outgoing transitions of a state have to sum up to one; including transitions with the same source and sink representing the system to stay in a state.
For some use cases, it might be useful to deal with continuous systems over discrete and even over continuous time. In that case, an agent can decide when to apply a certain action in the continuous time space $[0, \infty]$. Semi Markov decision processes cover continuous systems over continuous time. They model the systems dynamics as function over the continuous state space, allow the planner to apply an action whenever the system state changes and model the time spent in a certain state as probability function [171]. Although continuous system dynamics is mathematically expressed by a function over the continuous state space, it is usually numerically approximated by a large, discretized model.

As a result of applying action $a$ to state $s_{source}$ ending up in state $s_{sink}$ the agent receives a reward $r \in \mathbb{R}$ given by

$$r = R(s_{source}, a, s_{sink})$$

The reward model $R$ describes the immediate reward of a transition mapping it to the real numbers; W.l.o.g. costs are expressed as negative rewards. The reward value defines the relation between multiple objectives that are covered by (additive) state and action costs. An action $a_i$ with a higher reward than an action $a_j$ will be preferred by the agent with the ratio of the reward value and with respect to the current state. It is common to simplify the reward function depending on the successor state and the action:

$$r = R(s_{sink}, a)$$

If the reward not depends on the predecessor but to the successor state $s_{sink}$, it is calculated by [196]:

$$R(s_{sink}, a) = \sum_{s_{source} \in S} T(s_{sink}, a, s_{source}) R(s_{source}, a)$$  \hspace{1cm} (2.4)

As for the state and action space, the rewards are assumed to be stationary. Mapping the tuples of action and state to $\mathbb{R}$ enables comparing its value to develop strategies for non-myopic optimal control, since $\mathbb{R}$ provides a total order.

### 2.4.2 Control policy

MDPs allow modeling a controlled stochastic process with respect to immediate rewards. The element of selecting the action to choose for a certain state is called policy $\pi$; it represents the intelligence of the
agent to take decisions. \textit{(Intelligence is interpreted as taking decisions with respect to long-term effects subject to a cost function.)} Since the effect of actions are stochastic and resulting in different possible states, the optimal control policy cannot necessarily be represented as single sequence of actions. Instead, the policy is expressed of best applicable action for each state: $\pi : S \rightarrow A$. This implicates the policy to be independent of the history (Markov property). The proof by induction can be found in [196]. But due to the stochastic nature of the system, even applying a deterministic policy to the process can lead to different state trajectories. Nevertheless, one can find the optimal policy using planning methods (see Chapter 2.3).

2.4.3 Planning algorithms

As already introduced, \textit{planning} is the process of developing intelligent decisions (actions) for given situations (states). Planning algorithms are implemented as computer programs to automatically deduce control policies for system models w.r.t. to an optimality criterion (see below).

Finding the best actions can be done differently. A myopic approach would be to select the action with the best immediate reward for each state. Using \textit{planning} automatically create non-myopic, optimal control strategies.

The optimal policy - in terms of optimal reward - is deduced from simulating the system’s behavior. The MDP model describes the system dynamics under the effect of actions, their probabilistic outcome and the reward model expressing the immediate reward of an action. Instead of manually expressing the controller strategy, the planner develops the plan automatically from simulating the system model.

Calculating the system behavior over time allows the controller to compare actions based on their reward as cumulative sum of immediate rewards. The expected cumulative sum criteria defines the value function $V$ (also called utility function). It describes the expected utility of applying policy $\pi$ starting at state $s$: $V^\pi : S \rightarrow \mathbb{R}$, as will be introduced below.

Since the value function is mapping a policy for a state to $\mathbb{R}$, it enables comparing policies $\pi$. Formally, the goal of the planner is to find the optimal policy which is better than any other policy independently from the starting state $s$:

$$\forall \pi \in \Pi, s \in S : V^\pi(s) \leq V^{\pi^*}(s) \quad (2.5)$$

Since the value function defines implicitly the policy, the optimal policy can also be written as:

$$\pi^* = \underset{\pi \in \Pi}{\arg \max} V^\pi. \quad (2.6)$$

with $\arg \max_a \{f(a)\} := \{a|\forall y : f(y) \leq f(a)\}$.

The planner searches for the optimal policy evaluating $V$ with respect to an optimality criteria $\forall s \in S$ [171][196]. The finite horizon criterion $V^\pi_T(s) = E\left[\sum_{t=0}^{T-1} r_t | s_0 = s\right]$ accumulates the immediate reward up to a certain point in time $T$; it is motivated by the fact that the planner has to control the environment for the next $T$ steps only. For infinite horizon problems ($T \rightarrow \infty$) the discounted criterion $V^\pi_\gamma(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right]$ with $\gamma \in [0, 1)$ is motivated by the fact that events far in the future do have less impact onto the current decision. This causes the planner to prefer shorter plans and it assures the value iteration planning algorithm, presented below, to converge. The total reward criterion $V^\pi(s) = E\left[\sum_{t=0}^{\infty} r_t | s_0 = s\right]$ sums up the overall rewards; It is used when a finite horizon exists
but is not known exactly. And the average reward criterion \( V^\pi(s) = \lim_{T \to \infty} E \left[ \frac{1}{T} \sum_{t=0}^{T} r_t | s_0 = s \right] \) expresses the average future overall rewards of a policy.

Since the expected costs are additive, the 'Belman optimality principle' and the dynamic programming approach [49] can solve the problem. Based on that, the common techniques value iteration, policy iteration and Linear programming as well as combinations are used to solve MDPs [72] [184] [196]. In general, policy iteration should be preferred over value iteration for small MDPs.

**Value iteration**

Value iteration basically rolls out the system’s behavior from a given starting state \( s_{init} \) over time until reaching a stop criterion given by the optimality criteria. Since the utility of a state is given by the Bellman equation:

\[
V(s_{source}) = \max_{a \in A} \left\{ R(s_{source}, a) + \sum_{s_{sink} \in S} T(s_{source}, a, s_{sink}) V(s_{sink}) \right\} \quad (2.7)
\]

the value of a state can be recursively calculated over time through backward induction:

\[
V_{t-1}(s_{source}) = \max_{a \in A} \left\{ R(s_{source}, a) + \sum_{s_{sink} \in S} T(s_{source}, a, s_{sink}) V_t(s_{sink}) \right\} \quad \text{with } 0 \leq t < T \quad (2.8)
\]

After reaching the stop criterion, the value of the states at stage \( T \) is w.l.o.g. defined as maximal immediate reward \( V_T = \max_a r(s, a) \) without any further successor investigation. Starting from that leaf nodes, the values are propagated backward through the decision tree to the root node \( V(s) \), representing the value of starting state \( s \). Finally, the policy is deduced from the value function \( \forall s \in S \) with

\[
\pi(s) = \arg \max_{a \in A} \left\{ \sum_{s_{sink} \in S} T(s, a, s_{sink}) V(s_{sink}) \right\} \quad (2.9)
\]

Figure 2.4 shows the forward search process as an AND-OR-tree. The MDP states are represented as OR-node indicating the selection of one or another action. The AND-node represents the probabilistic transition into the successor states with respect to the action. After rolling out the future states up to horizon \( T \) (in Figure 2.4 it is \( T = 2 \)), the values of the leave nodes (states at stage \( T \)) are calculated and propagated back to the root node - the starting state \( s_{init} \) - as discussed above.

The resulting value function and policy converges to the optimal value function and policy for \( T \to \infty \). The most common way to stop the algorithm w.r.t. numerical inaccuracy is using a small number \( \epsilon \) instead of zero for measuring the evolution of the value function through an iteration: \( ||V_{t+1} - V_t|| < \epsilon \). The algorithm has a computation complexity of \( O((|S||A|)^T) \), since every action in \( A \) is applied to every state in \( S \) over the horizon \( T \). At every stage \( t \in 0, \ldots, T \) the value is evaluated as sum over all successor states and the optimal immediate reward from all actions.

**Policy iteration**

Policy iteration is an alternative to value iteration for infinite horizon problems [171]. In contrast to deducing the policy from evaluating the value of applying an action to a state, the second class of planning techniques, policy iteration, finds the optimal policy evolving a given policy: Starting with an arbitrary policy, it evaluates the value of the current policy on each state of the system (one step step backup) and
changes the policy according to the gathered value. The algorithm might start with an arbitrary policy $\pi_0$ and repeats until the policy will not be improved any further.

The discounted value evaluation ($\gamma \in [0,1)$) applies the current policy on the Bellman equation (see Equation 2.7) $\forall s_{source} \in S$:

$$V_{n+1}(s_{source}) = R(s, \pi_t(s_{source})) + \gamma \sum_{s_{sink} \in S} T(s_{source}, \pi_n(s_{source}, s_{sink}))V_n(s_{sink})$$ (2.10)

The policy improvement is done by extracting the policy from one-step backup on the value function:

$$\pi_{n+1}(s_{source}) = \arg \max_{a \in A} \left\{ R(s_{source}, a) + \sum_{s_{sink} \in S} T(s_{source}, a, s_{sink})V_n(s_{sink}) \right\}$$ (2.11)

Solving complex systems or large-scale spaces to approximate continuous systems is computational expensive and time consuming. Hence, simplifications are an important tool to handle MDP of realistic size.

## 2.5 Partially observable Markov decision processeses

Partially Observable Markov Decision Processeses (POMDPs) combine the idea of Markov Decision Processes (see Section 2.4) and Hidden Markov Models (HMMs) to consider measurement errors. To express the uncertainty of sensor information, HMMs provide a framework to estimate the state of a dynamic system (more general a random variable) based on uncertain observations. HMMs model the hidden system dynamics as Markov Chain. The occurrence of observations depends on the system state, which is modeled as probability distribution over the observation space for each state. With knowledge about the system dynamics and the measurement model, the estimator tries to infer the real system state from a sequence of observations, as shown in Figure 2.5.

Formally, a discrete HMM extends a discrete Markov Chain that models a random process with a state space $S$ and a state transition function $T$, by a measurement model. The extension defines a set of
observations $\Theta$ and the observation model $O : S \times \Theta \rightarrow [0, 1]$. Each element of the observation model denotes the probability of the system being in the hidden state $s$ when measuring observation $o$. It is modeled as conditional probability $P(s|o) = O(s, o)$. If, additionally, the previously applied action also affects the observation, the model is extended to $O : S \times \Theta \times A \rightarrow [0, 1]$ given by $O(s, o, a) = P(s|o,a)$.

Like Bayesian estimators, the POMDP observation model does not make any assumptions on the class of the probability distribution.

### 2.5.1 Bayesian estimator

The goal of recursive estimation is to gather the actual system state from uncertain measurements within a continuous loop over time. Uncertain measurements cover incorrect or inaccurate measures, which could stem from ambiguities.

To overcome incorrect measures, the estimator utilizes models that describe the dynamics of the observed system and/or the correlation between measurement values and system values. (The correlation models uncertainty.) At each recursive update step, the estimator updates an internal representation of its belief (belief state) about the current system state based on a measurement and the models. Eventually, the estimate of the actual system state, which is a single state, is deduced as final result from the belief.

It is important to note that state tracker (or estimator) do not cover controlled systems, such as MDPs, although system control can be directly deduced from the estimate. A state tracker does not consider the effect of actions onto the system.

Similar to the definition of the MDP in Section 2.4, the system dynamics are given by the transition function $T(s_{sink}, s_{source}) = P(s_{sink}|s_{source})$ and the measurement model is given by $O(s, o) = P(s|o)$.

A HMM based Bayesian state tracker recursively updates the belief state $b$ that is modeled as probability distribution over the state space. It expresses the probability of the system being in the state, respectively. For discrete systems with finite state space, the belief is expressed as discrete probability distribution over the state space. Formally, the belief state $b$ is an element of the continuous belief space $\mathbb{B}$:

$$\mathbb{B}^{|S|} = \left\{ b \in [0, 1]^{|S|} \mid \sum_{i=0}^{|S|-1} b_i = 1 \land b_i \geq 0 \right\}$$

as probability distribution over the states $s_i \in S \quad i \in \{0, \ldots, |S| - 1\}$. The dimension of the belief
space is the number of states of the underlying MDP process $|S|$. And $b$ is a probability vector; each element represents the probability of the system being in the corresponding state:

$$b = (P(s_0), P(s_1), \ldots, P(s_{|S|-1}))$$

(2.13)

The probability of a MDP state $s_i$ ($P(s_i)$) for belief $b$ is denoted as $b(s_i)$, $b_i$, or $P(s_i|b)$.

The recursive belief update is defined for step $t$ as follows. Measuring an observation $o_t$ at stage $t$, the tracker updates its current belief $b_t$ (a probability distribution over the state space) component-wise by the following function $\kappa : \mathbb{B} \times O \rightarrow \mathbb{B}$, $\forall s, s' \in S$:

$$b_{t+1}^{o_t}(s') = \kappa(b_t, o_t)(s)$$

$$= P(s'|o_t) \sum_{s_{sink} \in S} P(s_{sink}|s) \cdot b_t(s)$$

$$= O(s', o_t) \sum_{s_{sink} \in S} T(s, s_{sink}) \cdot b_t(s)$$

(2.14)

First it applies the transition function of the hidden Markov process to the current belief $b_t$ and then it adds the knowledge of the observation $o_t$ in order to get the new belief state $b_{t+1}$ with respect to the observation model and a normalization factor $\eta$. In a second step, the tracker estimates a single system state $\hat{s} \in S$, which can be done with several mathematic estimation methods; for instance maximum-likelihood:

$$\hat{s} = \arg \max_{s \in S} \{b(s)\}$$

(2.15)

As a result, the tracker filters the system state from a sequence of uncertain measurements up to stage $t$. Like for MCs and MDPs, the Markov property holds for Markov process based state trackers: The belief only depends on the last belief and the update function $\kappa$.

### Simplification using specific probability distributions

Another, widely used version of the Bayesian state tracker is the Kalman Filter [117]. It assumes the observation model and the process model to be Gaussian distributed, which is motivated by the common nature of sensor noise and actuator errors. As a direct consequence, the filter only needs to handle the two scalar values of mean and variance instead of the belief state vector with cardinality $|S|$; this significantly reduces the computational effort and memory usage.

In contrast, Markov Model based state trackers do not make such assumptions. Being able to deal with arbitrary probability distribution is necessary if the modeled probability distribution is built up as convolution of different distributions.

### Optimal control under uncertainty

The presented state tracker estimates the system state based on uncertain measures (observation model) and a process model that describes the system dynamics.
The system dynamics of controlled systems, however, are modeled with respect to the controller’s actions. Extending the state tracker process update model with respect to actions would allow tracking and controlling a process under uncertainty. The state tracker update function $\kappa: \mathbb{B} \times \mathbb{O} \rightarrow \mathbb{B}$ (see Equation 2.14) is extended to $\tau: \mathbb{B} \times \mathbb{O} \times \mathbb{A} \rightarrow \mathbb{B}$ taking into account the effect of the applied action $a$ to the transition. Thus, the belief state update equation 2.14 is extended as follows $\forall s \in S$:

$$b_{t+1}^{o,a}(s') = \tau(b, o, a)(s) = P(s'|o_t, a) \sum_{s_{sink} \in S} P(s_{sink}|s, a) \cdot b_t(s)$$

$$= O(s, o_t, a) \sum_{s_{sink} \in S} T(s, a, s_{sink}) \cdot b_t(s)$$

(2.16)

The action $a$, that is a) applied to the system and b) used to update the belief state, is deduced from a policy $\pi$ as introduced before.

To apply a MDP policy $\pi(s)$ as presented in Section 2.4, the belief state is transformed to a MDP state $s$ using for example the maximum likelihood estimator. This extends the framework for model predictive control with the ability to deal with sensor noise. Although the policy takes into account the uncertainty of the effect of actions onto the probabilistic system, it does not consider the fact of uncertain measurements (e.g. sensor noise). This is - so far - covered by the state tracker separately. Separating the process update and the sensor update is valid for discrete [206] and continuous [221] linear stochastic regulator problems. Linear stochastic regulators formulate optimal control of a linear, stochastic process. If, however, the control input is constraint, the optimal control can not be deduced by setting the differential to zero, like widely applied in Linear Quadratic Regulators. In the case of restricted control, finding the optimal control is a planning problem as introduced in Section 2.3.

The resulting value function $V_{MDP-ML}: \mathbb{B} \rightarrow \mathbb{R}$ is defined in Equation 2.17 according to the MDP value function $V_{MDP}$

$$V_{MDP-ML}(b) = V_{MDP}\left(\arg \max_{s \in S}\{b(s)\}\right)$$

(2.17)

Such a value function is shown in Figure 2.6. It shows a value function for a 2-state MDP $(s1,s2)$ with two actions $(a1,a2)$. The belief space on the horizontal axis is defined according to the definition in Equation 2.12. The values of the $<\text{state,action}>$ pairs are assigned as dots. To calculate the value for belief states $b$ it is mapped onto the MDP state space through the maximum likelihood function, as defined in Equation 2.15. (The uniform belief state $(u = (\frac{1}{2}, \frac{1}{2}))$ is mapped to $s1$ by definition.) The implicit policy for MDPs with maximum likelihood estimator is defined equally to Equation 2.11.

Although this approach enables applying limited control to uncertain situations, the control law is only optimal for states that are certainly known to the controller. If, however, the state estimate is wrong, the policy is likely to return an unreasonable action [182]. The control is not robust. In contrast, considering the uncertainty of measuring the system state in the planning process results in optimal policies under uncertainty. A framework for such problems is POMDPs. The method combines the field of estimating and controlling stochastic processes to planning under uncertainty. Unique is that the planner takes into account the uncertainty of measurement in developing the strategy. POMDPs provide a framework for
modeling control problems with partial observability in terms of uncertain action effects and incomplete knowledge of the resulting system dynamics with respect to multiple objectives. Hence, a POMDP describes a discrete combinatorial decision problem with uncertainty in decision outcomes and partially observable system state.

Since the belief is a probability distribution over the belief space, which is developed according to the underlying MDP, the POMDP can also be seen as MDP over the belief space. Therefore, POMDPs are also called belief state MDPs.

### 2.5.2 Formal model definition

Stationary POMDPs can be described as a six tuple $(S, A, \Theta, T, O, R)$ defined as follows:

- $S = \{s_1, \ldots, s_{|S|}\}$ is the state space of the dynamic system
- $A = \{a_1, \ldots, a_{|A|}\}$ is the space of actions applicable to the system
- $\Theta = \{o_1, \ldots, o_{|\Theta|}\}$ is the set of observations from the system
- A set of state transition probabilities given by the function $T : S \times A \times S \rightarrow [0, 1]$
- A set of observation probabilities given by the function $O : S \times A \times \Theta \rightarrow [0, 1]$
- A reward function given by $R : S \times A \times O \rightarrow \mathbb{R}$

The system is in one state at any point in time, which is not directly observable by the agent. This is modeled by a finite set of observations with occurrence probabilities conditioned by agent actions and system states. The agent can execute and transition probabilities for successor states conditioned on predecessor belief state and action. A reward function maps action-state pairs to reward values and a discount factor relates these values over different time steps.

The planner’s goal is to select the optimal correction action based on previously accumulated information about the system’s state. This accumulated information is represented as a probability distribution over all possible system states (referred to as the belief state of the planner/agent). Representing all information available to the agent at a point in time, a belief state fulfills the Markov Property [196]. An action choice is optimal if it results in a maximal expected value over all possible futures of the process.
The POMDP extends the MDP definition, as described in 2.4.1, by a finite set of observations and the observation model. As introduced in the context of HMMs, the observation model describes the probability of getting an observation $o$ given the system to be in state $s$: $P(o|s)$. Similar to the transitions for finite systems, the observation probability can be expressed as multi-dimensional stochastic matrix representing the probability $P(s_j|o_i, a_k)$ of the system being in the $j$’th state after getting the $i$’th observation after applying the $k$’th action. The matrix for a given action $a$, as shown in Equation 2.18 is applied in the state tracker to develop the belief state.

$$O_a = \begin{pmatrix}
P(s_0|o_0, a) & \cdots & P(s_{|S|-1}|o_0, a) \\
P(s_0|o_1, a) & \cdots & P(s_{|S|-1}|o_1, a) \\
\vdots & \ddots & \vdots \\
P(s_0|o_{|\Theta|-1}, a) & \cdots & P(s_{|S|-1}|o_{|\Theta|-1}, a)
\end{pmatrix}$$

(2.18)

Note that the POMDP formulation in this thesis only considers stationary systems for simplicity reasons. That is, the POMDP components do not change over time. A POMDP additionally may define algorithm parameters such as the discount factor $\gamma \in [0, 1)$, which assures the value iteration to converge (see optimality criteria in Section 2.4.3), and the initial belief state $b_0$ of the planning algorithm. Given the POMDP definition, the controller has to select the most rewarding action to be executed in the current situation.

**2.5.3 Optimal control policy**

As for MDPs, the optimal policy can be expressed as the action that maximizes the accumulated future reward with respect to an optimality criteria (see Section 2.4.3).

The immediate reward of a belief state $b$ is given by the sum of MDP state rewards, weighted by the probability of the state, respectively:

$$R(b, a) = \sum_{s \in S} R(s, a) b(s)$$

(2.19)

With respect to the independence of transitions, rewards and observations, the optimal value function is generally given by:

$$V^*(b) = \max_{a \in A} \left\{ R(b, a) + \sum_{o \in \Theta} P(o|b, a) V^*(\hat{\tau}(b, a, o)) \right\}$$

(2.20)

with developing the belief state under a given observation and action by $\hat{\tau}$ as an extension of $\tau$ defined in Equation 2.16 (see Chapter 2.5.1).

$$b^a_{t+1}(s) = \hat{\tau}(b, o, a)(s) = P(s|o_t, a) \sum_{s_{sink} \in S} P(s_{sink}|s, a) \cdot b_t(s)$$

(2.21)
2.5 Partially observable Markov decision processeses

Then, the optimal policy for belief \( b \in B \) returns the actions with the best value given by \( \pi^* : B \rightarrow A \):

\[
\pi^*(b) = \arg \max_{a \in A} \left\{ \sum_{s \in S} R(s, a) P(s|b) + \sum_{o \in \Theta} P(o|b, a) V^*(\hat{\tau}(b, a, o)) \right\}
\]  \hspace{1cm} (2.22)

Finite horizon policies of the length \( T \) can be expressed as policy tree with the height \( T \). Starting at a certain state in stage 1, the agent can select a policy and gets an observation recursively until the horizon \( H \) is reached. Since the number of actions and observations are finite, each finite horizon policy can be expressed as policy tree. As shown in Figure 2.7, each node can be labeled with an action, so that the actions within the control loop (sequence of applying an action and getting on observation) are defined.

A recursive definition of the policy tree is provided by conditional plans. A conditional plan \( \gamma \in \Gamma \) is a tuple \(< a, \beta >\) of an action \( a \in A \) and an observation strategy \( \beta : O \rightarrow \Gamma \). A particular conditional plan defines which plan to follow after applying action \( a \) with respect to the received observation. At stage \( H \), the conditional plan is just an action. For a given belief \( b \), the finite horizon policies is expressed a conditional plan [170] [196], mapping the sequence of observations to conditional plans. Since the number of trajectories grows exponentially in the number of observations and actions, conditional plans can only represent the policy for a short horizon, although one could use the effect of history cycles and express the conditional plan as state machine.

As shown in Section 2.4, a policy maps the state space to the best action if the Markov property is fulfilled. This holds for POMDPs since the transition, observations and rewards satisfy the Markov property. As a consequence of the belief space, the strategy for POMDP is expressed as function from the continuous belief space to the action space: \( \pi : B \rightarrow A \). \( \pi(b) \) denotes the action to be executed for belief state \( b \).

**Piecewise linear and convex representation**

As introduced for MDPs, the value function provides a method to assess and compare policies based on different optimality criteria introduced in Section 2.4.3: total reward, discounted reward[201], average reward or reward up to certain point in the future (finite horizon criteria) [197].
The value of a conditional plan \( \gamma = (a, \beta) \) with respect to total reward criteria is defined as:

\[
V^\gamma_T(s) = R(s, a) \\
V^\gamma_t(s) = R(s, a) + \sum_{s' \in S} T(s', a, s) \sum_{o \in \Theta} O(s', a, o)V^\beta_{t+1}(s')
\]

The value for a belief state \( b \) with respect to the conditional plan \( \gamma \) is calculated as follows:

\[
V^\gamma(b) = \sum_{s \in S} b(s)V^\gamma(s)
\]

and the optimal value for \( b \) is given by:

\[
V^*(b) = \max_{\gamma \in \Gamma} \left\{ \sum_{s \in S} b(s)V^\gamma(s) \right\}
\]

Thus, similar to the MDP definition, the value functions define a partial order over the policy space \( \Pi \).

Two policies \( \pi \) and \( \pi' \) are comparable by its value \( V \) for a belief state \( b \):

\[
\forall b \in B, \pi \in \Pi : V^\pi(b) \geq V^\pi'(b)
\]

Equation (2.25) shows that the value of a policy is linear in the belief and Equation (2.26) shows that the optimal value for a finite horizon is simply the upper surface of a set of linear functions. Hence, the optimal value function for POMDPs with finite horizon is piecewise linear and convex [197].

Let \( \alpha \) be a vector of the size \( |S| \) whose entries represent the values of the conditional plan starting with a certain action \( \gamma \) for all states \( s_i \in S \) and let \( \Xi \) be the set of all finite policy \( \alpha \)-vectors:

\[
\alpha = (V^\gamma(s_0), V^\gamma(s_1), \ldots, V^\gamma(s_{|S|-1}))
\]

Then the optimal value for belief \( b \) can be rewritten as:

\[
V^*(b) = \max_{\gamma \in \Gamma} \left\{ \sum_{s \in S} b(s)V^\gamma(s) \right\} = \max_{\alpha \in \Xi} \left\{ \sum_{s \in S} b(s)\alpha(s) \right\}
\]

Hence, the value function can be represented by a set of tuples that consists of an \( \alpha \)-vector and the corresponding action \( < \alpha, a > \). This set is sufficient to allow the controller to select the optimal action for the current belief by performing a greedy one-step lookahead:

\[
\pi(b) = \arg \max_{a \in A} \left\{ \sum_{s \in S} R(s, a)b(s) + \sum_{o \in \Theta} P(o|b, a) \max_{\alpha \in \Xi} \left\{ \sum_{s \in S} b(s)\alpha(s) \right\} \right\}
\]

A piecewise linear and convex value function for a two-state POMDP with five actions is illustrated in Figure 2.8(a). The belief space expresses the probability of being in state \( P(s = s_1) = 1 - P(s = s_2) \). The optimal value function is given by the upper surface as combination \( \alpha \)-vectors. Hence, the optimal value (and policy) for belief state \( b \) is given by a linear lookup over the set of \( \alpha \)-Vectors:

\[
\pi^*(b) = \arg \max_{a \in A} \left\{ \sum_{s \in S} b(s)\alpha_a(s) \right\}
\]
2.5 Partially observable Markov decision processeses

Point cloud representation

Due to Pineau et al. [163], alternatively, the value function is represented as a point cloud of pairs \( <b, V(b)> \) of a belief state and its value \( V(b) \) as shown in Figure 2.8(b). The belief space is covered by linearly interpolating the value function between the sampling points [163]. The lookup is linear in the number of sampling points, since the neighborhood in terms of the convex hull of the requested point is needed to interpolate its value. Such a approximating only yields for upper bounds, due to the convexity of the value function.

Grid-based value function approximations vary in set of the sampling points and the interpolation method [200][163]. A survey of point-based POMDP solvers is given by Shani et al. [193].

2.5.4 Automated planning

As described in Section 2.4.3, planning is the process of automatically finding the optimal control policy, given by Equation 2.30. It is also called solving the given problem. However, numerical limitations lead to an approximation of the optimal policy. Solving this for POMDPs is very similar to MDP problems, since a POMDP is a MDP over the infinite Belief Space, if the observations can be assumed as independent. Hence, solving POMDPs builds on planning MDPs (see Chapter 2.4.3).

The optimal value function in Equation (2.20) can be approximated using dynamic programming. As for MDPs [171] the following recursive equation converges to the unique optimal value function:

\[
V_{t-1}(b) = \max_{a \in A} \left\{ \sum_{s \in S} R(s,a)(s) + \gamma \sum_{a \in \Theta} \sum_{s \in S} P(o|s,a)(s) V_t(\tau(b,a,o)) \right\} \quad \text{with} \quad \gamma \in [0, 1) \tag{2.31}
\]

To calculate the value for a belief state, the algorithms rolls out the future behavior up to a certain point \( T \), calculates the value of the leave nodes given by the best immediate reward \( V_T(b) = \max_{a \in A} R(s,a)b(s) \) and propagates the value through the planning tree up to the root node. The planning tree can be represented as And-Or-tree. In contrast to the MDP representation, the nodes are belief states instead of system states \( s \in S \). The edges from a belief-state-node represent actions, which lead - with respect to the transition function - to an intermediate-belief-state-node. The edges from the intermediate-belief-states represent the outcome for the observations ending in belief states.
To approximate the optimal value function with a certain error $\epsilon$, iterative deepening processes the algorithms explained above increasing the search depth until the improvement of the value in iteration $i$ is small: $||V_i(b) - V_{i-1}(b)|| < \epsilon$, which is guaranteed due to the discount factor $\gamma$.

The value iteration algorithm improves the value for a single belief state. To come up with a value function that covers the whole belief space, point based methods evaluate a set of sampling points and linearly interpolate the value function in between, as shown in Figure 2.8(b). The algorithm is similarly used to evaluate the $\alpha$-vector representation of the value function. After calculating the more detailed values, existing entries may be useless. Removing redundant entries is important since the computational complexity of a policy lookup is linear in the number of entries. Hence, developing the value function needs a mechanism to prune vectors or points that do not effect the value function any more.

### 2.5.5 Control under uncertainty

As discussed in Section 2.4, MDP control enables deducing non-myopic optimal control for systems with limited control options, that have uncertain effects onto the controlled system. MDP control is optimal if the system state is known. If the system state can only be observed through uncertain measurements, Bayesian estimation allows deducing a reasonable system state. However, the control policy might not be optimal, since the planner relies on the real system state, which might not be estimated correctly.

POMDP controller cover uncertain observations and action effects in planning. Intuitively, the advantage of using POMDPs is that the control policy is capable of preferring actions that reduce the uncertainty about the system state instead of the optimal expected reward. Formally, this is expressed by the reward calculation of the belief state (see Equation 2.19). Each MDP state reward is weighted by the state probability of the belief, which represents the controller’s certainty about the system being in that state.

The difference of a value function for a MDP controller with maximum likelihood estimator and a POMDP is shown in Figure 2.9, for a two state MDP with two (three) actions. It shows $\alpha$-Vector based value functions for POMDPs (see Section 2.5.3) and for a MDP controller with maximum likelihood estimator, as introduced in Section 2.5.1. The maximum value, and therefore the policy, differs for the two controllers, indicated by the gray areas. In Figure 2.9(a) the POMDP value function defines the value of action $a_2$ higher even on the left side of the uniform belief, where the state $s_1$ is more likely with optimal action $a_1$. The reason for that depends on the relation between the values in state $s_2$. In $s_2$ the value of action $a_1$ is, compared to the value for $a_2$, worse than it is vice versa for state $s_1$. The POMDP considers the level of certainty of the system’s state: The belief’s value is a combination of the belief state probability ($b(s)$) and the reward values of the corresponding MDP state $R(s, a)$, which - in this case - leads to a different value and a different control policy.

If, however, a third action $a_3$ is additionally considered, which is pairwise dominated for the MDP states $s_1, s_2$, it will not be part of the MDP policy. As shown in Figure 2.9(b), it might nevertheless be best for uncertain areas of the belief space.

### 2.5.6 Approximations for large systems

The process of evaluating the value function for POMDPs is computational expensive. Due to Roy et al. [182], this ‘[…] is the reason why solving POMDPs is considered to be intractable’. Hence, real world problems with thousands of states are not feasible with naive algorithms and value function representations. However, approximative methods might allow finding useful solutions, under careful consideration of the problem.
Policies of large POMDPs and MDPs are memory expensive. The exact value function contains a value for every state. To simplify the representation of the value function, it can be approximated from a finite set of generative functions on Splines, Fourier or Trigonometric bases. The benefit of this approach is that a generated function covers an arbitrary complex state space being expressed only by a limited set of parameters. The drawback is the value function and the deduced policy are not necessarily optimal due to the projection error. This approach is applicable to the common value iteration and the policy iteration algorithm. After updating the value function with the Bellman equation, the value function is projected to the closest generative function according to a norm, which is used as value function from there on. Common regression methods like the least square method or linear regression can be used to find the closed representation from various generative functions.

Computing the optimal policy exactly is memory expensive and time consuming for realistic, large-scale systems. Due to the \( \max \) operator, the value equation (see Equation 2.7) is nonlinear and finding the optimal MDP policy can not be solved efficiently using linear algebra techniques. Instead, iterative algorithms are used. Although the basic mathematic operations are simple, the algorithms must deal with large vector sets in case of large action and state spaces of practical problems. Parallelizing the calculations is an appropriate method to speed up the policy calculation. Approximative algorithms unadulterately calculate the policy reducing the computational effort by decreasing the accuracy of the value function calculation. This is motivated by the fact that if the value function approximation is close to the correct value function, the policies are almost equal as well.

For instance, conditional planning by planning out the future up to a finite horizon approximates the value of a state and the optimal policy for the infinite horizon. Another example is the discounted reward criterion. It approximates the optimal value function depending on the discount factor \( \gamma \), as introduced in Section 2.5.4. Although, it is technically used to let the planning algorithm converge for infinite horizon problems, it is intuitively motivated by modeling the decreasing impact of the immediate reward of future actions to the overall reward with respect to the process in time.

As introduced in Section 3.3, simplifications, such as Belief space compression and online planning are options to reduce the computational effort of solving POMDPs.
2.6 Reinforcement learning

‘Reinforcement learning is the problem faced by an agent that must learn behavior through trial and error interactions with a dynamic environment’, due to Kaelbling et al. [116]. Inspired by behaviorist psychology, it is a research area of computer science that investigates how agents take actions in an (partly) unknown dynamic environment in order to maximize some notion of cumulative reward. Hence, it is closely related to MDPs and POMDPs. In particular, it extends the class of tackled problems.

To automatically deduce the optimal control policy from evaluating the given system model, automated planning, as discussed in Section 2.5.4, assumes a model of the controlled system. If, however, the model is not known in advance or the controller shall adapt to a changing system, the reinforcement learning paradigm describes how to learn to act in an unknown environment. In general, it is tried to answer the questions of how a controller can learn from observations of unknown, partially observable environments to act optimally, that is, to maximize their expected reward.

Generally, a learner can develop a policy by challenging the system. It tries a certain action and gets the resulting cost and state (or observation) in return from the system. Hence, the learner does not have the model of the probabilistic system dynamics, it can not use it as presented above. Instead, it approximates the system by sampling. The average error, which is the difference between the learners information about the system and the real system, decreases with the number of samples. Although adapting the policy to the real system causes the controller to apply non-optimal policies, it enables to continuously improve the incomplete and imperfect model and deal with system changes.

Usually, learning controllers act on real systems which causes real effects. This requires the learner to balance the control policy between learning from the environment (exploration) and applying its already learned approximately optimal control (exploitation). However, learners can also be applied to real world simulations in order to deduce a policy approximatively.

Model-free reinforcement learning methods learn the policy in a specific value function similar to the MDP planning value function, which was introduced in Section 2.4.3, by sampling the system. The learner remembers the rewards it has received for applying an action in a certain situation (system state). Since the learner remembers the accumulated rewards, the resulting policy is strategically optimal. Learning algorithms differ in terms of the strategy to deduce the optimal policy from sampling the system (convergence rate of learning). It balances between 1) exploiting the already experienced effect of actions (greedy select the action which is so far known to be best), and 2) exploring unknown effects of new actions (uniformly select action) with the risk of getting less reward.

Technically, the $\epsilon$-greedy learning policy is parametrized by a the scalar variable $\epsilon \in [0, 1]$. The most rewarding (i.e. exploiting) action is used with probability $1 - \epsilon$ and a random action (i.e. exploring) is selected with probability $\epsilon$.

This balance is important if the system learns the policy for an unknown environment online, since bad rewards do effect the agent. The simulated trajectories are either defined randomly (Monte Carlo Method) or searched directed with the temporal difference methods like sarsa [215] and q-learning [216] for Markovian problems or heuristic search [160].

In particular, q-learning approximates the estimated value of a state-action-tuple $(s, a)$ with $s \in S, a \in A$ by the q-value function

$$Q : S \times A \rightarrow \mathbb{R}$$

This function represents the expected costs for applying an action $a$ to a state $s$, similar to the value
function as a result of planning. Hence, the according policy is given by

$$\pi(s) = \arg\max_{a \in A} \{Q(s, a)\}$$

The value function is learned by sampling the system. Similar to planning methods, the value update (see Equation 2.32) is a composition of the immediate reward for applying an action \(a\) to the current system state \(s_t\), which is given as \(R(s_t, a)\) from the sampled system, and the value of the successor state and optimal action, which is given from the already learned values, as \(\max_a Q(s_{t+1}, a)\). According to the Bellman equation, the future reward is weighted with a discount factor \(\gamma \in [0, 1]\) in order to balance the score between immediate and future reward. The learning is stabilized by an exponential smoother with parameter \(\alpha \in [0, 1)\)). The value update function is shown in Equation 2.32.

$$Q(s_t, a) \leftarrow (1 - \alpha) \cdot Q(s_t, a) + \alpha \cdot \left( R(s_t, a) + \gamma \cdot \max_{a \in A} \{Q(s_{t+1}, a)\} \right)$$  \hspace{1cm} (2.32)

Model-free reinforcement learning algorithms develop a policy of the system without knowledge about the dynamics of the underlying system. Instead of merely remembering the rewards it has received, model-based reinforcement learning also profit from remembering the sequence of states the system has entered. This information is used to learn a model of the dynamics of the system. Remembering not only rewards but also state and action processes enables to apply planning techniques (see Section 2.4.3) to improve the learning strategy. Introductions and further references to different methods of learning are available in [196].

Common methods rely on full observability gathering the system state and the reward. Learning to act in an unknown and partially observable environment, however, is a difficult variant of reinforcement learning. Partially observability leads to the effect of perceptual aliasing [65], where the same observation is gathered in distinct states where different actions are required. To learn in partially observable domains, reinforcement learning is combined with planning under uncertainty. Like in the domain of fully observable problems, this problem is solved either by model-free policy deduction [144] or model-based variants learning the Hidden Markov Model [53] and deducing the policy from POMDP planning. A survey of model-free and model-based for learning under uncertainty is presented by Shani et al. [190]. However, common learning methods require a strict separation between learning and application phase. In [191], Shani combines the two phases presenting an online approach for model-based POMDP learning.

A model-based approach is learning Hidden Markov Models. It can be done with variants of the Baum-Welch procedure [47], i.e. [48], a special case of the Expectation–maximization algorithm. To learn POMDPs, which can be seen as HMM that describe a controlled process, the method is extended to learn HMM for each control action.

However, the main issue of model-based learning under partially observability is the size of the model in terms of state and observation space and the large number of samples to deduce the probabilistic model.

2.7 Summary

A Markov decision process provides a framework to model a stochastic dynamic system under the effects of a limited set of controller actions (see Section 2.4). Modeling a problem as MDP enables a controller to automatically develop a plan of how to act in certain situations. Considering the short-term and long-term
effects of actions allows developing non-myopic strategies. Approximation methods enable to deal with large-scale problems and reinforcement learning methods allow, in contrast to planning, approximating the optimal control policy with imperfect knowledge about the system model.

Markov decision processes can deal with uncertainty in terms of the action effect onto the controlled system. This framework assumes that the current situation - the system state - is known to the controller. Hence MDPs are also called *Fully Observable Markov Decision Processes*. In fact, perfect measures are mostly not the case, due to inaccurate and incomplete measures stemming from physical limitations and the abstraction of the modeled aspects.

To deal with measurement uncertainty, a separate (Bayesian) filter can be applied. It delivers an estimate (a concrete state) to the MDP controller, based on uncertain measurement. Hence, a strategic MDP controller, is capable to handle uncertain system information. This estimating controller does not know the system state. Instead, it tracks a belief state as probability distribution over the possible system states, which is updated by incoming observation (observation model) and applied actions (transition model). Both contain uncertainty. Nevertheless, the MDP policy is not necessarily optimal, due to the estimation error.

Considering not only the uncertainty of actions effects but also the measurement error in developing the optimal control strategy, leads to POMDPs. POMDPs provide a model-based framework to optimally control dynamic systems under uncertain effects of actions and uncertain sensor information about the system (see Section 2.5). Like MDPs, POMDPs allow deducing a control policy with respect to uncertain, temporal aspects of the controlled system. In contrast to MDPs, where the controller has full access to the current system state, the POMDP controller holds a belief state to express its level certainty about the system state. Moreover, the control policy is deduced with respect to uncertainty, which makes planning complex. Due to the resulting computational effort, however, the calculation of POMDP policies generally is not feasible for real world problems. As for MDPs, the basic principle of planning algorithms for POMDP is dynamic programming. To reduce the complexity, POMDPs are modeled factorial and planning algorithms find approximating solutions [195].

Planning, however, requires a model of the controlled system to deduce an optimal policy. If the model is not known in advance or the controller must be capable to adapt to a changing system, *Reinforcement learning* (see Section 2.6) enables learning to act in an *unknown, partially observable* environment. In general, a learner develops a policy by challenging the system. The underlying learning strategy enables to efficiently deduce a control policy. Although adapting the policy to the real system causes the controller to apply non-optimal policies, it enables to continuously improve the incomplete and imperfect model and deal with system changes.

Usually, learning controllers act on real systems which causes real effects. This requires the learner to balance the control policy between learning from the environment (exploration) and applying its already learned approximately optimal control (exploitation). However, learner can also be applied to real world simulations in order to deduce an approximately optimal policy efficiently.
Related work

Markov decision processes (MDPs) provide a framework for sequential planning problems under uncertainty about the controlled system dynamics. Their success comes from the development of efficient algorithms for finding optimal policies [171]. The basic principles, i.e. the formal model definition and the planning algorithms for solving MDPs, are described in Section 2.4.

Markov decision processes, a particular form of sequential decision problems that fulfill the Markov property, have been successfully implemented in various fields of application. One of the earliest fields of application of sequential decision processes where inventory problems [171] deducing optimal re-ordering strategies of goods for a production system. Queuing theory [100], a branch of operations research, for example, models systems of waiting lines to infer the delay of single items passing the system. In this case, actions are 1) scheduling the items and 2) the path selection. With respect to a reward model which expresses the optimization criteria, an MDP planner automatically deduces the optimal scheduling and routing strategy. Queuing theory is also applied in traffic engineering, production engineering, factory design and many more domains. Discrete event simulation is one implementation.

MDPs describe controlled processes with uncertainty in terms of the effect of the control action onto the system dynamics. However, MDPs assume that the system state is known to the controller (full observability). The framework does not consider the fact of uncertainty that stems from observing the system state by sensors that cause error and/or ambiguity.

In general, measurement models express the mapping from physical aspects of the real world to informations (measurement values). A specific case of this mapping is uncertainty which stems from calibration or discretization. It affects the optimal control, since the control law relies on the real system state but the controller can not recognize it. This problem is usually resolved using estimators (filters). In the context of Markov processes, Hidden Markov models solve a temporal estimation problem. Formally, a hidden Markov model (HMM) probabilistically describes a Markov process with partially observable states. HMMs estimate the system state from uncertain observations based on the knowledge of the underlying process model. Initially developed as method for statistical inference for probabilistic functions over finite state that fulfill the Markov property [48], HMMs are widely applied in recognition of temporal patterns (or sequences) in natural language processing such as handwriting [56, 109], gesture [154, 218], and speech recognition [172].

Instead of setting up HMMs manually, the model can be learned from measurement data. In most of the applications, the model topology (number of states and observations) is manually determined in advance and the model parameters are estimated by an expectation–maximization algorithm [84] such as the Baum-Welch algorithm [47]. Ford proposes an alternative learning method [86].

Estimating such an uncontrolled process on temporal bases can be extended with control actions that effect the states transitions. The process model can be treated separately without loss of optimality if
the continuous models are stochastic and linear, repetitively [221]. However, considering uncertainty in deducing the optimal strategy for discrete MDPs is formulated as Partially observable Markov decision processes.

POMDPs extend MDPs allowing more uncertainty in terms of a measurement model. They combine MDPs with HMMs to describe that the controller does not get the correct system state but rather an observation that correlates with the real system state [185]. As stated in Section 2.5, the model complexity, however, increases the computational effort of the algorithms to compute the optimal control policy. Similar to MDPs, the success of POMDPs depends on efficient algorithms.

Although the majority of POMDP models are developed in the research areas of artificial intelligence instead of control, the number of applications continuously grow, due to the progress which is made in solving POMDPs. The improvements range from approximative solution, model compression [182] to heuristic search [97] which does not only consider the mathematical or computational aspect of the planning problem, but gains from application domain specific characteristics. Hence, the development of the algorithms profits from the growing field of applications and vice versa.

### 3.1 Applications

Today, POMDPs are rarely used in industrial applications, since the problem definition must either be small enough in order to be solvable, which might end as trivial problem, or the solving algorithm can be adjusted to the problem structure such that the policy can be calculated with realistic effort.

#### 3.1.1 Industrial applications

Cassandra [62] provides an overview of POMDP applications: Industrial applications are machine maintenance and structural inspection. In machine maintenance, the controller should come up with a strategy for periodic inspection and repair of a machine, due to loss of adjustment or attrition, with respect to maximizing the production capacity [197][162]. Agrawal et al. [32] discuss stochastically deorientation with partial observability for multiple machines.

A similar problem is structural inspection. Instead of maintaining a machine, the inspection of structure, i.e. bridges, buildings, is controlled optimally with respect to uncertainty [79]. Controlling an elevator is also a control problem under uncertainty, since the optimal control depends on the number of passengers and their designated target, which is not observable. Instead, the controller only gets the information that at least one passenger wants to enter the elevator at a certain level. Fei et al. [82] and He et al. [99] present a POMDP-based selection policy for sensors as part of a large sensor network in order to provide optimal coverage. A theoretical consideration of the control limit for a two-state POMDP, that describes a replacement problem as simple production process with the state to be either 'good' or 'bad' and actions 'continue' and 'replace', is shown in by Grosfeld-Nir [93]. Online planning and scheduling is applied to control a modular printer as presented by Ruml et al. [183].

#### 3.1.2 Autonomous systems

Due to Thrun et al. [208], autonomous or semi-autonomous control of robots is a challenging field of applications for POMDPs. The robot is used to access hazardous environments, but it has, in any case, a limited view to its surrounding through its sensors. The modeled states are the location, surrounding aspects (i.e. temperature), and internal state, i.e. battery level. The actions are the robots actuators and
3.2 Solving POMDPs

Observations are sensor values. The reward is set according to the robot’s goals. In case of a deep-ocean exploration, which does not allow controlling the robot remotely, the control actions to move the robot are uncertain due to unexpected current. Probabilistic planning for robotic exploration is presented by Smith [198]. Unmanned aircraft collision avoidance is expressed as POMDP in [41, 220].

On the basis of game theory, POMDP are extended from single to multi-agent systems by methods of decentralized partially observable Markov decision process. This aspect is described in detail by Sigaud [196].

3.1.3 Computer vision

Decision process models are also applied to computer vision problems: Visual attention is presented by Bandera et al. [43]. To reduce the effort of video processing, a decision maker selects the region of interest with a high resolution which is surrounded by a large area with low resolution. Montero [150] presents a video conference system that selects the best view from gestures that indicate the user’s activity. The proposition combines gesture recognition with a POMDP model to select the best view. Hoey [101] presents a method for learning decision theoretic models of human behaviors from video data to enable value-directed human behavior analysis which can be used for intelligent video surveillance.

3.1.4 Assistant systems

In the 1970es, Smallwood and Sondrik [197] propose the application of human learning and instruction. From the AI’s perspective, they define the hidden state as the status of knowledge, observations as the responses to queries and alternative presentation of material. Today, assistant systems guide the human user, i.e. personal assistants shown by Varakantham et al. [213], organize tasks as presented by Myers et al. [152] or help people with dementia introduced by Hoey et al. [104]. The guidance relies on interacting with the human which is, at least, recognizing the user’s intention by filtering from sensing his actions (see gesture detection with hidden Markov models). Further improved, intelligent assistants are additionally capable to predict the user’s goals, which enables getting prepared, i.e. wheelchair navigation [207]. A comprehensive framework for intelligent assistants is presented by Fern [83] and Hoey et al. [102].

A fundamental component of assistant systems is the human-machine-interface. Since natural language processing has a temporal character, a POMDP-based framework for spoken dialogue management is presented by Young et al. [223] and Williams et al. [217].

3.2 Solving POMDPs

The fundamentals of solving POMDP are introduced in Section 2.5.4. Value iteration develop the value of a belief state. The corresponding policy is deduced by applying a one-step backup. Whereas policy iteration directly develops the policy. In the following, common offline planning techniques are summarized.
3.2.1 Solution methods

POMDP policy-iteration is presented by Smallwood and Sondik [197] as well as Hansen [96]. Value-based algorithmic methods for solving POMDPs for a finite horizon or a belief space grid are presented in [137]. Point-based value iteration is presented by Pineau et al. [163]. It approximates the value function by computing the values for a finite set of belief states (sampling points) and interpolates in between them. However, the selection of the sampling point is crucial. It is for example the set of reachable beliefs. The software Perseus randomly explores the belief space according to the given POMDP model and stores the accessed belief space [202]. Then it builds the new value function (set of \(\alpha\)-vectors) from either one-step-backups values or the old value function [193]. However, it quickly converges to an approximate solution for medium-size problems. Heuristic Search Value Iteration controls the trajectory of evaluating the belief space. It maintains a lower and an upper bound of the value function. The distance allows to measure the quality of the approximation and is used as search heuristic (selecting the next belief states for investigation) [199]. Instead of the additional upper bound, the Forward Search Value Iteration uses MDP traversal information to control the POMDP belief state investigation [192]. Both algorithms maintain a growing number of \(\alpha\)-vectors. Since the resulting policy is deduced by evaluating the value of each vector, it is crucial - even for the controller - to remove vectors that never represent the maximum value of a belief state. This process is called \textit{pruning} [193].

3.2.2 Solver software and toolkits

The following section summarizes the existing POMDP solvers and toolkits.

Murphy provides an MDP toolbox for Matlab [7] that supports the basic value and policy iteration algorithms for discrete MDPs.

The Perseus algorithm developed by Spaan and Vlassis is a set of Matlab functions implementing a randomized point-based approximate value iteration algorithm to solve flat POMDP [12]. Symbolic Perseus is an extension of Perseus to also solve factored POMDPs, written in Java by Poupart [11] to solve not only flat but also factored POMDPs. Also based on Perseus is an extension written by Porta; it allows solving POMDPs for continuous state spaces [165, 166, 167]. It uses algebraic decision diagrams (ADDs) for representing POMDPs.

SPUDD also uses ADDs to represent value functions and policies [103]. It implements a value iteration algorithm for MDPs and POMDPs [21].

The MultiAgent decision process toolbox is a C++ software toolbox for decision-theoretic planning and learning in multi-agent systems [22]. Approximate POMDP Planning Software is a C++ implementation of the SARSOP algorithm [41] for solving discrete POMDPs. The 'pomdp-solve' program [23] written by Cassandra solves flat POMDPs based on linear programming solvers in C. The solver uses dynamic programming for finite and infinite horizon problems with or without discounting. It stops after calculating the value \(\epsilon\)-optimal with various stopping conditions. The implemented solution algorithms are Enumeration, Two pass [197], Linear Support [137], Witness, Incremental Pruning [61] and Finite Grid as instance of Point-based Value Iteration [163]. The solver is based on linear programming libraries. It supports either the public domain Mixed Integer Linear Programming software 'lp_solve' [6] or CPLEX [3].

His proposed file format for representing POMDP models and policies (\(\alpha\)-vectors and policy graphs) is widely used. Smith provides the The ZMDP software package on [25]. It reads Cassandra’s model files and implements several heuristic search algorithms. The output policy is expressed in a JSON format. A
POMDP controller framework for Python is written by Stollmann and Migge [14]. It provides reading POMDP models and policies in a simplified form of Cassandra’s and the zMDPs format. A file format preprocessor - Cassno - is written by Stollmann and Migge in Haskell [2].

The RL-POMDP software [19] is a reinforcement learning algorithm that finds approximate solutions of POMDP problems.

### 3.3 Relaxations

Instead of solving the POMDP completely, approximative solutions try to relax the problem and reduce the computational effort. Compressing the belief space is an example for simplification on modeling the problem and online planning is a relaxation on the search algorithm.

#### 3.3.1 Belief space simplification

**Belief state augmentation** is an approach to reduce the computational complexity of POMDP planning. Instead of representing the belief state as probability vector with cardinality \(|S|\), which allows modeling arbitrary modes, the distribution is limited to a certain family of distribution functions, i.e. Gaussian distribution. This approach reduces the complexity of the belief state propagation from manipulating a discrete probability distribution (a vector of size \(|S|\)) to the parameter set to describe the distribution, which is two scalars (mean and variance) for a Gaussian distribution. This is limited to distribution that are closed to multiplication, such as Gaussian distributions.

Automated belief space compression, as presented by Roy et al. [182], reduces the cardinality of the belief space with respect to representing a set of sampled belief states. This is motivated by the assumption that a controller does not necessarily reach every possible belief state and, hence, calculating the policy over the entire belief space is not needed. Thus, the policy is only calculated for the sub-space of the belief state samples, which reduces the computational effort. Although the solution is approximating the optimal policy under uncertainty, it outperforms the policy by an MDP controller with estimator for robot navigation tasks [182].

In contrast to the common flat model representation of MDPs, factored MDPs utilize the structure of the given problem to compactly represent large MDPs [54]. The complex state space is modeled using state variables and the transitions are modeled using a dynamic Bayesian network. The transitions can be compactly expressed if the transition only depends on a small number of state variables (dimensions). And the immediate reward can also be decomposed of rewards from small subsets of state variables. The explicit representation allows a reduction in representing the problem, but increases the complexity of solution algorithms. Solution algorithms for factored MDPs are summarized in [95].

**Online planning** and **learning** enhance the framework by enabling to adapt the control law to the environment. Instead of deducing it directly from a complete system description, which might be not available for complex systems, it allows to approximate the optimal control policy from challenging the environment or a simulation while the controller is already applied.

#### 3.3.2 Online planning

Usually, the controller strategy is pre-computed offline on an a-priori defined model and afterwards applied as static lookup table in the controller, which is cost effective. **Offline planning** is time and
memory consuming, since the policy needs to be valid globally.

In contrast, online planning develops the control policy while the controller is already in use. Given a model, the online planner focuses on evaluating the current state, which is given by the controller during runtime. Hence, the computational effort of planning is reduced by focussing on the occurring states. As a result of evaluating only the occurring states, the policy is developed for the relevant states only. Although the developed policy is not optimal for all states, like in classical offline planning, this might not be needed since the system stays in a certain region of the state space.

Since online techniques approximate the optimal policy online, the controller can adapt to system changes, e.g. reinforcement learning.

The time for online planning is limited to the control loop frequency, since the current belief state is updated in each iteration step. The policy calculation is limited in time by observing the system state and applying the next action. This brings up a new optimality criteria for planning algorithms (see Section 2.4.3). Similar to the finite horizon criteria, online search is forward reasoning over a finite horizon starting at the current state. Instead of limiting the planning iteration by a fixed horizon, the forward search is limited by time. It uses Iterative Deepening to deepen the search step-wise [184].

Online planning algorithms for POMDPs are summarized by Ross and Paquet [181]. The search is optimized by using heuristics to control the search direction for dynamic programming - real-time dynamic programming - [46]. Instead of expanding the search uniformly, only the best action from the current policy is investigated. The LAO* algorithm [96][97] improves policy iteration by heuristic search on a graph instead a planning tree, reusing the evaluated states.

Although the investigated search space is reduced to the local search around the current state and search heuristics thin out the search paths, online planning is still computational expensive, since the error of the approximated policy gets small only for a large search depth (horizon). Thus, offline and online calculations are combined: Since the horizon of the online simulation for the current state is limited by the time given by the principle of the control loop, the online controller benefits from accessing the value of the leave notes, which is calculated offline.

Due to the limited computation resources, the core question of online planning is the balance between the depth and the width of investigating the system by forward search and the balance between applying actions to either exploit the already gathered knowledge or explore the system (see Chapter 2.6), in terms of selecting the start state of the iteration.

3.4 Research gap

This chapter shows the current state of the art of intelligent agents, in particular strategic control under uncertainty, is shown. In particular, the results of research in the field of MDPs, HMMs, POMDPs and Reinforcement learning are shown. Then example applications in the field of industry, autonomous systems, computer vision and assistant system give an overview of how POMDPs are applied to real world problems today. However, the work commonly lacks for concrete models and process definitions of how to develop models.

In general, intelligent agents interact with their environment to reach a certain goal. How they act in a certain situation is internally defined as control strategy. Sequential decision making allows using control strategies that take into account long-term effects in terms of multi-objective costs, that depends on the system state and control actions. Stochastic models take additionally into account the uncertain affect of actions onto the controlled system (MDPs) and uncertain measures of the current system states.
3.4 Research gap

(POMDPs) in terms of observations. Hence, the controller considers not only the system but also the quality of knowledge about the system. In both cases, automated planning software enables automatic deduction of long-term optimal strategies from models.

Handling that many aspects leads to complex models and high calculation efforts in automated planning. The first question in applying POMDPs is: Which characteristics of a problem are required to justify applying a complex and expensive method of POMDPs?

As shown in this chapter, solving POMDPs is a well-studied field. Several frameworks and algorithms are developed in C, C++, Java and MATLAB in order to effectively deduce optimal control policies. Relaxations, such as belief state simplification and online planning significantly reduce the effort of automated planning. To compare POMDP solver algorithms, a set of small and mid-scale standard problems are commonly used, which can be formulated directly as a POMDP. But for real-world problems, the current implementation approaches face a dilemma. Fast solvers are written in C or C++. The development of complex models requires high-level modeling environments, like MATLAB/Simulink®. But solvers in MATLAB are not as efficient as in C or C++. Hence, the question is: How to deduce strategic control policies under uncertainty for large, real-world problems?

This can be tackled from two sides: To reduce the effort, either the planning algorithms are improved (solving domain) or the problems are expressed as small models (problem domain), without losing its statement.

The application side approaches try to model given problem in a way, that existing solver software can handle it in a reasonable amount of time. The solver side approaches the problem from the opposite direction. Independently from the problem, it develops software to effectively deduce plans from given models. However, the bridge between the two fields is the modeling language.

Industrial applications are large and complex by definition. Thus, the upcoming question approaching POMDP from the problem domain side is: How to model real-world problems, such that a strategic control policy can be automatically deduced?

The modeling process is not investigated in great detail. Once a problem is formally defined in an appropriate modeling language, which is - in case of POMDPs - commonly unintuitive and causes errors, solver software automatically deduces a control law. Hence, developing solvers is encapsulated from the problem domain by the model definition. And developing the model bridges the gap between grasping a given problem in its specific domain and expressing it as a POMDP model. However, this requires a deep understanding in the problem domain and the solving domain, in terms of the methodology of POMDPs.

Specific problems, such as the parallax error correction on interactive screens, see Chapter 5, require controller extensions. As will be discussed in detail in Chapter 5, upcoming question in applying POMDP are: How to extend an estimation to consider a multi-objective cost function? And, how to deal with event driven observations in recursive estimation and control?
Controller design

Chapter 3 introduced dealing with MDPs, HMMs and POMDPs from the algorithmic point of view. Besides the development of algorithms to solve POMDPs efficiently, expressing the problems is an effective approach to apply POMDPs to real world problems.

The following Chapter focuses on modeling POMDPs and extends the framework theoretically. First, it lists the problem characteristics that make POMDPs necessary and shows simplifications and the benefit of using automated planning under uncertainty. Then, several process of modeling POMDPs is described. It shows the dependencies of the single model elements, it discusses how to manually express them and it presents how to automatically deduce non-myopic control strategies from measurement data or existing simulations.

Additionally, two theoretical framework extension are presented. A novel principle of a POMDP controlled estimate is presented that enables to express recursive estimation problems with respect to a multi-objective cost function. Further, an observation model expressing the oblivion process for a Bayesian Filter is presented. The convergence of the introduced observation model is proven as well as the parameter for setting up the convergence speed towards the uniform belief state.

4.1 Problem characteristics

POMDPs are a powerful but complex framework for strategic control under uncertainty that are modeled probabilistically. However, probabilistic models are unintuitive, hard to express and cannot be verified intuitively. Moreover, probabilistic models cause high computational effort in automatically deducing the control policy with respect to the temporal nature of the given problem. Hence, the effort of expressing the problem and designing the controller must not outweigh the benefit of using the controller. This is, to profit from optimal strategic control of a dynamic system under uncertainty.

4.1.1 Aspects

To assure that the given problem can not be covered by simpler, less complex control method, it must fulfill certain characteristics. Although POMDPs technically cover simpler problems, i.e. full observability, it is worth applying simpler control methods (i.e. MDPs) to solve the problem with less computational effort. If, however, even the action affects the system with certainty, even simpler methods (i.e. Linear programming) sufficiently solve the given problem.

In general, POMDPs express sequential decision problems of controlled systems with respect to uncertainty in terms of uncertain measures and uncertain action effects and a cost function. The problem
characteristics that make POMDPs necessary are listed below. If one of the aspects is trivial to express, i.e. the observation model is a bijective mapping between observations and state, it might not be necessary and a POMDP might not be the right method for the given problem.

If, for example, the control can immediately compensate the system state, the problem is no sequential decision process. It can be optimally solved with a ‘greedy’ policy and the long-term temporal aspects of the problems are not relevant.

The following paragraph lists the problem characteristics to involve POMDPs. If one of the aspects is not given, a simple method might be applied. Such simplifications are described in the proximate Section 4.1.2.

**System state**  The system is in a state at each time. The system states describe the relevant aspects of the given problem. The system state can be modeled as a variable over a multi-dimensional continuous space or a finite (and discrete) set. The belief state of the POMDP represents a probability distribution over the states. Hence, the latter case is simply expressed as discrete probability vector $b$, without restrictions to the mode of the distribution. Continuous probability functions, however, are more complex to express without any mode restrictions.

**Observations**  The system state is partially observable. Observations provide limited information about the actual state of the system, i.e. ambiguities (perceptual aliasing), noisy or faulty measurements. A probabilistic mapping from observations to the system states expresses the measurement model.

**Actions**  The system is controllable. This is that actions effect the system state. Moreover, the effect is of stochastic nature: The actions effect the system state under uncertainty. And the control actions are limited, which requires a sequential decision process.

**Dynamic system**  The system dynamics are relevant. The system evolves over time. It is expressed as a process with a structure. This is, for example, that the system is transferred into an enclosing subspace of the state space. The system dynamics are modeled as state transitions.

**Sequential decision process**  The controller captures an observation and applies an action at each control loop step. Due to the nature of machine control, it is reasonable to assume discrete time steps. For simplicity reasons, the model expresses discrete, uniform time elapsing.

**Cost function**  The preference of the controller is expressed as cost function. Rewards are expressed as negative costs. The cost function assigns each state, action, observation tuple an immediate cost value, which might be zero. The cost function is used as optimality criterion by the planner. It aims to develop a control policy that optimizes the control with respect to optimizing the cost function over time.

**Markov property**  The state transition, the observation and the reward function are assumed to be Markovian. This is, that the successor only depends on the current situation not on the past trajectory. If this does not hold, the underlying elements may be consolidated.
4.1 Problem characteristics

Time horizon  POMDPs consider optimal control over time. Hence, a horizon is either given (finite time optimal control), or, for an infinite horizon problem, a discount factor defines the decreasing influence of the accumulated costs (evaluation) over time. See optimality criteria in Section 2.4.3.

Finite spaces  Although the states actions and observations can be finite or continuous in general, in the following they are assumed to be finite for simplicity reasons, due to the increasing complexity of the belief state of the POMDP.

I/O Interface  The topology of the POMDP model represents symbols of observations, actions and states due to abstraction from the real world. Symbolic actions and observations represent the interface between controller and controlled system. The sensors and actuators of the controller, however, deal with measurement world values instead of symbols. Hence, an I/O interface maps the real world values to the index of the controllers internal symbolic representation, as shown in Figure 4.1. Moreover, multiple dimensions of the underlying real world spaces must be projected onto the one-dimensional POMDP model space.

The mapping of the I/O Interface is the core reason for large POMDP models. POMDP models are based on symbolic states, observations and actions. Due to the mapping of multiple dimension spaces and discretization, the POMDP topology gets large.

4.1.2 Simplifications

The model aspects, as described above, define a POMDP. However, the resulting model would be highly complex. If the state and action space are continuous, the transition function would be defined as probability function over continuous spaces. The same holds true for the observation model. However, this leads to piecewise closed formulas which are complex and hard to design. A continuous state space requires a POMDP controller to manage a belief state as continuous probability function, which is difficult to manage unless the model is not limited to a specific family of distribution.

Thus, states, actions and observations are assumed to be a finite set. A finite state spaces enables the controller to handle the belief state as arbitrary discrete probability distribution. Finite actions and observations allow modeling the transitions and observations as probability matrix and enable the planner to simulate one step of the system dynamics by investigating the finite Cartesian product of transitions and observations given by the actions and observations.

Considering the system to be stationary with a constant time step size significantly reduces the model size, since transitions and observations are modeled independently of the system stage (time).

If the system is fully observable, this is the system state is accessible by the controller, the problem can be reduced to an MDPs. And even if the system is not fully observable, the planning problem can be
treated as a fully observable MDP with a separate estimator that covers uncertain measurements.

4.1.3 Automated planning under uncertainty

A greedy policy is, in general, not optimal. Reasons for that are limited control action, absorbing situation (Absorbing subsets of the state space do not allow the system to escape to any other state outside the subset.) or the fact that a immediately worse action might lead the system into a better state (w.r.t. the cost function) in the long run.

In other words, the controller can not optimize the system without considering its temporal aspects. Strategic control, however, develops control policies that are optimal over time. The process of deducing a control policy for a POMDP problem is as follows. Given a POMDP model of a system, a planner automatically deduces an optimal strategic policy. Eventually, the policy is integrated into a controller.

Automated planning is a great benefit for optimal control. Only the immediate effects (within one time step) in terms of dynamics and rewards are modeled. The planner then automatically deduces a strategic control law that is optimal over time. This might be hard to realize rationally for humans.

Planning deduces a control policy from investigating the system dynamics that fulfill the Markov property. In case of dependencies on the past, the chain can be augmented in a more abstract Markov process. To simulate the system behavior and cost over time, a horizon or a discount factor must be given to assure the simulation to converge. The specifications of a POMDP (states, actions, observations, transitions, observations-model and cost function) are mandatory for planning.

POMDP planning combines non-myopic optimal control with uncertainty. Hence, the controller weights up its certainty of current system’s state estimate and the cost (reward) of a control action to find the optimal control action. This is a big benefit of planning under uncertainty. However, POMDP rely on a good model of the given problem.

4.2 Modeling the problem

As stated in Section 4.1.3, automatically deducing strategic control policies relies on a definition of the given control problem, in particular a model. The formal POMDP model elements, as introduced in Section 2.5.2, are divided into two groups: the topology and the functions of the system. The topology models the system’s aspects in terms of state, observation and action. Each aspect defines dimensions and discretization within the dimension. A dimension defines an aspect of the system that is considered in the model (i.e. the temperature or the position of the controlled system).

POMDP (and MDPs) are either expressed flat (explicit or extensional) or factorial [54]. In flat modeling, each value of each dimension is enumerated directly. Hence, the resulting topology models the cross product of all values over all dimensions of the system characteristics. This can hardly be modeled intuitively. Factorial modeling, however, expresses only certain features of the system. A state is fully defined by assigning a value to each feature.

In general, the functions of the system are mathematical functions from the Cartesian products of the topology spaces onto a probability space or, in case of the reward model, $\mathbb{R}$. The functions define the system dynamics (transition model), the observations model and the rewards. Since the functions are mostly modeled by probability matrices of the cardinality of the topology, the models get huge are not intuitive to handle.

The dependencies between the POMDP model parameters are shown in Figure 4.2. The capabilities
of the controlled system define the observation space and action space in terms of dimension. The discretization of the observation space influences the state space, since a state is necessary only if it is indicated by an observation, although observations might indicate ambiguities.

The reward model defines the immediate reward function with respect to the controller’s goal from the problem definition. By definition, the reward function depends on states and actions. The observation model defines the pairwise correlation between each observation and state (action)-tuple. Hence, it depends on the state, and observation (and action) space in terms of dimension and discretization. The same holds true for the transition model describing the system dynamics. It depends on the state space (sink and source state of the transition) and the action, which effects the system dynamics.

However, for non-trivial modeling, the action, observation and state spaces grow quickly. Since large models are hard to handle manually, automatically deducing the (PO)MDP model from more intuitive models, that are expressed in a more intuitive language, as presented in Section 4.2.2.

The assumption that the model will not change over time (non-stationary process) does not always apply. Instead of deducing a POMDP model, the controller can implicitly learn the control policy while planning, as shown in Figure 4.4(c). Although this approach reduces the effort of modeling the problem, learning the policy does converge over time to the optimal policy, which might lead to applying non-optimal control to a real world system. To reduce the risk of applying non-optimal control to an production plant, learning the policy is implemented on querying a simulation.
4 Controller design

(a) Passive model deduction

(b) Interactive model deduction

(c) Policy learning

Figure 4.4: Automated modeling process

Figure 4.5: Simulation system interface

4.2.1 Manual modeling

Manual modeling, as shown in Figure 4.4(a), is the process of manually define the POMDP model. Every aspect of the POMDP model is formulated manually. This gives the engineer complete freedom in expressing the problem. The main issue, however, is the unintuitive nature of expressing the single model aspects, i.e. large probabilistic matrices, which carry a high potential of making errors in the modeling process. A second difficulty are the dependencies of the model elements, as shown in Figure 4.2. If, for example, the observation space is changed, it effects almost all other elements of the POMDP model.

Automatic sanity checks, like check if summing up the probability values equals 1, help modeling the POMDP manually.

Instead of manually expressing the POMDP model, it can be deduced automatically from measurement series or a simulation of the system.

4.2.2 Automated model deduction

Instead of manually modeling, the POMDP model can be deduced automatically from measurement series of a system. In general, a controlled system can be expressed as function $S : S \times A \rightarrow S \times R \times O$, as shown in Figure 4.5. Figure 4.4 shows the three automated modeling approaches that are discussed in
the following. This approach reduces the effort of modeling the problem and is more robust to modeling errors.

If state, observations and actions are accessible and the topology is given, the transition model and the observation model can be automatically deduced from counting the occurrences. The system outcome is counted and translated into probabilistic models. By the law of large numbers, the resulting distributions approximate the system. If the topology is not given, it can also be automatically build up on occurrence for a given discretization. Even if the state is not directly observable, training algorithms, i.e. the Baum-Welch procedure [47], automatically infer a Markov model.

If, however, the system state is not directly accessible to the modeler, the expectation-maximization algorithm [71] is used to set up a hidden Markov model. Based on a series of observations and a given model topology, the algorithm sets up a process and observation model of a Markovian system. Originally, the Algorithm obtains an uncontrolled Markov model with hidden states. It is extended to controlled processes by setting up a Hidden Markov model for each action, which is admissible due to the Markov property. Automatic reduction of the model topology is shown by Stolcke et al. [203]. Hence, it is possible to deduce a POMDP model from a system by sampling a time series of (action, observation) tuples. The measurements might either be sampled from a real system or an existing simulation.

The process chain of deducing a POMDP model from measurement data is shown in Figure 4.4(a). The probabilities for the transition- and the observation model are deduced by counting occurrences. However, defining the topology of the model, must be done manually. In order to end up with a high quality transition- and the observation model, the topology might be selected with respect to the representativity of the samples.

How the topology of the POMDP model (action space, observation space and state space) is set up accordingly to the measurement series is shown by example. In Chapter 5, such a POMDP model is deduced from conducting offline experiments for correcting the parallax error on interactive screens. A user study directly measures the system dynamics (movement of the user viewpoint) and the measurement model (correlation between observations and system states). The example application shows, how the topology of the POMDP model is set up accordingly to the measurement series. In any case, the Markov property is assumed to hold true.

Deducing a POMDP model from measurement is a passive process, since the Extractor can not influence the incoming sample data, which might be useful in order to set up good models.

Instead, the POMDP model can also automatically be deduced from challenging a given simulation of the controlled system. Interactively deducing a POMDP model from a simulation is shown in Figure 4.4(b). Although sampling the system causes computation effort, this allows to express the given problem in a more intuitive, high-level language, such as MATLAB/Simulink®. This approach allows applying strategic control, in particular MDPs and POMDP, to complex problems. Since the high-level language allows engineers expressing the system dynamics and deducing optimal control without being familiar with POMDP details.

In order to model the physical nature of continuous systems the state space gets large to not loose information due to discretization. A large state space, however, also causes the need of a large set of training data. Hence, automatically deducing POMDPs for complex system from data causes large expenses. Moreover, automatically deducing the control law (planning) additionally causes computations costs. To reduce the extraction costs and increase the quality of the resulting model, the extractor can follow a sampling strategy. However, due to the separation of the planning- and extraction-process, the extraction strategy can not benefit from planning results.
4 Controller design

To avoid the high costs for extracting the model and planning the policy separately, both processes can be combined. By merging the steps, the extraction process can utilize the results of the planning process in order to select the next sample.

Instead of setting up the model first and deducing the policy afterwards, Reinforcement learning techniques are developed to automatically deduce control policies from challenging the controlled system in order to adapt to a changing system. Reinforcement learning extends the bare process of developing the control policy by parametrized learning. The parameter define the strategy of challenging the real system since the results of action immediately effect the controller. If, for example, the learning controller has already gathered information about the system it can decide to exploit its information in order to positively effect itself or explore the system in order to gather new information about the actions effects. This, however, does only make sense, if the controller is immediately effected by actions, which is for example the case if a robot explores unknown territory.

4.3 Multi-objective estimation

As introduced in Section 2.5.1, recursive estimation problems generally track a system state based on uncertain observations. The estimation is recursively changed according to the gathered observation and the process model. However, the estimate update is commonly not provided with any costs, i.e. the recursive Bayesian estimator [38]. The effect of changing the estimate in contrast to a wrong estimate might cause different costs. Large changes might cause high costs, while small changes cause less costs. And a great offset between the actual system state and the estimate might cause higher costs than a small estimation error. The controller which updates the estimate sequentially gathers observations that provide evidence for the quality of the current estimate, respectively the offset between the current estimate and the actual system state. To consider the longterm effect of the actions with respect to uncertainty, updating an estimate can be formulated as sequential decision problem, i.e. a POMDP (This work arose from discussions with Tim Schmidt.). The uncertainty of the action’s effect represents the uncertainty of the controlled estimate and the uncertainty of the systems feedback is covered by uncertain observations.

The proposed framework extends common estimator with a cost function that represents multiple objectives: First, the cost of a wrong estimate and second the cost of changing the estimate.

The parallax error correction on interactive screens, described in Section 5, is an estimation problem with respect to such costs. The parallax error is corrected based on the user’s viewpoint which is estimated from uncertain interaction errors. The controller actions change the viewpoint estimate after getting evidence if the current estimate is correct. Changing the viewpoint estimate, however, directly effects the parallax correction on the screen. Hence, the control action directly depends on the estimate and it is changed if the estimate is changed. Although an interaction might not be corrected perfectly the user might be more irritated by the fact that the correction changed. Hence, it is reasonable to confronting costs for the current estimate and changing the estimate.

Formally, the estimation controller is defined as follows. As shown in Figure 4.6, the controlled estimator gathers measurements \( m \) from the system and returns an estimate \( e \) about a hidden system state. Given the current estimate, the measurement is translated into and observation \( o \) that indicates the error of the estimate (preprocessor). It is the offset between the current estimate and the measurement, which contains uncertainty due to the measurement error. The POMDP controller internally maintains a belief state that represents the offset between the current estimate and the actual system state with respect to uncertainty. Hence, the observations provide evidence about the controller states. The belief is updated according to a process model of the system state and an observation model (as introduced in Section...
2.5.1). The strategic policy defines a control action \( a \), which represents adapting the estimate (Estimator), according to the controller’s goal. This is the (internal) state that represents no offset between actual and estimated system state. However, the cost function represents two objectives: The state cost represents the cost for the estimation error and the action costs represent the cost for actually changing the estimate. Hence, the estimate is recursively updated strategically by a controller according to measurements with respect to a multi-objective cost function.

Another example is energy efficient targeting. If a machine controls ‘target acquisition’ with respect to energy efficiency, the controller that maintains the estimate of the target has two objectives. First, it should aim for the target correctly and second, it should not run out of battery. Any change of the estimate causes a correction of the aim and consumes energy. Hence, this estimation problem is effected by multiple objectives and might be covered by the proposed estimation controller.

### 4.4 Modeling oblivion

Discrete time controllers run in a scheduled loop according to the time steps of the system model. In the control loop, the belief state - a probability distribution over the system states - is updated by an observation (model) and the process model.

It is usually assumed that a controller queries its sensors. Such pull-based measurement systems gather an observation on a pull request, i.e. a thermometer provides the temperature to the controller if requested. For event-based measurement systems like GUI systems (see Chapter 5), the measurement is pushed to the controller. This is not critical if the update frequency of the events is higher than the control loop, since the events can be stored in a buffer of length one, which is filled by the sensor and pulled from the controller. If the event frequency is lower than the control loop update, the controller must deal with getting no observation.

If no observation is gathered, it is reasonable to develop the belief state according to the process model. In the following section the general principle of a null-observation is introduced. Without a process model it develops the belief state stepwise towards the uniform belief state, which represents no knowledge about the system state.
The novel oblivion process is mathematically described, proved to converge against the uniform belief state. The convergence speed is can be parametrized by a scalar value, which is also proven to be correct.

The intuitive purpose of the null observation is to express losing knowledge of the current situation. This is implemented by developing the controller’s current belief state $b_t$ as defined in Section 2.5.1 in Equation 2.12, towards the uniform belief state $u$ as shown in Equation 4.1. It represents not having any knowledge about the estimated system state, since every state of the state space $S$ is equally probable.

$$ u = \left( \frac{1}{|S|}, \ldots, \frac{1}{|S|} \right) \quad (4.1) $$

The process of forgetting is modeled as time-independent Markov process:

$$ b_{t+1} = P_k b_t $$

It is based on the single matrix $P_k$.

**Definition 1.**

$$ P_k = \begin{pmatrix}
  k & \frac{1-k}{n-1} & \cdots & \frac{1-k}{n-1} \\
  \frac{1-k}{n-1} & k & \cdots & \frac{1-k}{n-1} \\
  \vdots & \ddots & \ddots & \vdots \\
  \frac{1-k}{n-1} & \cdots & \frac{1-k}{n-1} & k \\
\end{pmatrix} \quad k \in \left[ \frac{1}{n}, 1 \right) $$

It is claimed, that matrix $P_k$ lets an arbitrary belief state $b$ converge against $u$ by multiple applications, while $k$ sets the level of convergence speed:

$$ \lim_{t \to \infty} P_k^{(t)} b = u $$

In general, a stochastic matrix, that describes a time-homogeneous process over a finite state space, converges to a unique stationary vector $v$ if the Markov chain is irreducible and aperiodic: $\lim_{t \to \infty} P^{(t)} b = v$. Hence, in the following it is shown, that the process is irreducible and aperiodic.

**Definition 2.** A Markov chain that describes the process $\{X_t\}_{t \in \mathbb{N}}$ is said to be irreducible if there exists a path to access any state $s_j$ from any state $s_i$. A state $s_j$ is said to be accessible from a state $s_i$, if a system started in state $s_i$ has a certain probability of transitioning into state $s_j$ after finite time:

**Proposition 1.** $P_k$ is irreducible.

$$ \exists m \in \mathbb{N} : P(X_m = s_j | X_0 = s_i) = (P_k)^{(m)}_{i,j} > 0 \quad (4.2) $$

**Proof.** This is given for $m = 1$, since $\forall i,j \quad (P_k)_{i,j} > 0$ by definition:
4.4 Modeling oblivion

\[ k \in \left[ \frac{1}{n}, 1 \right) \]
\[ \Rightarrow 1 - k > \frac{1}{n} \]
\[ \Rightarrow \frac{1 - k}{n - 1} > 0 \]  
(4.3)

**Definition 3.** The period of state \( s_i \) is defined to be the integer \( l \in \mathbb{N} \) such that \( p_{i,i}^{(n)} = 0 \) for all values \( n \) other than a multiple of \( k \). If \( l = 1 \), then the state is said to be aperiodic [100]. A Markov chain is aperiodic if every state is aperiodic [100].

**Proposition 2.** \( P_k \) is aperiodic.

**Proof.** Aperiodicity is given for all states with \( l = 1 \). Returning to state \( i \) can occur at irregular times, since it is possible to reach all sink states \( s_j \) from all source states \( s_i \) within one step of the process given by \( P_k \), since all values are non-zero by definition.

\[ \forall i : P_{i,i} = k > 0 \land \forall i, j \neq i : P_{i,j} = \frac{1 - k}{n - 1} > 0 \]  
(4.4)

**Definition 4.** A Markov chain is ergodic, if all states are ergodic. Recurrent and aperiodic states are called ergodic. 'A state is said to be recurrent state if, upon entering this state, the process definitively will return to this state again. Therefore, a state is recurrent if and only if it is not transient' [100]. 'A state is said to be a transient state if, upon entering this state, the process might never return to this state again. Therefore, state \( i \) is transient if and only if there exists a [different] state \( j (j \neq i) \) that is accessible from state \( i \) but not vice versa, that is, state \( i \) is not accessible from state \( j \)' [100].

**Proposition 3.** \( P_k \) is ergodic.

**Proof.** Since \( P_k \) is strictly positive, all states \( j \) are accessible from all states \( i \), not transient but recurrent. Since all states are furthermore aperiodic the Markov chain defined by \( P_k \) is ergodic.

Due to Hiller and Lieberman [100] holds the following. For any irreducible ergodic Markov chain, \( \lim_{t \to \infty} P_{i,j}^{(t)} \) exists and is independent of the start-state \( i \). Furthermore the stationary distribution is given by \( \lim_{t \to \infty} P_{i,j}^{(t)} = \pi_j > 0 \) and \( \sum_{j=0}^{S} \pi_j = 1 \) and it holds \( \pi P_k = \pi \) and \( \pi \) is unique [100].

In the following, it is shown that \( P_k \) has the stationary distribution \( u \).

**Proposition 4.** Since, \( P_k \) is irreducible and ergodic, it converges to the unique stationary vector \( u = \left( \frac{1}{n}, \ldots, \frac{1}{n} \right) \) with \( n = |S| \) \( (\lim_{t \to \infty} P_k^{(t)} b = u) \). And it holds \( u = P_k u \).
Proof.\

\[
\begin{align*}
\mathbf{uP}_k &= \left(\frac{1}{n}, \ldots, \frac{1}{n}\right) \cdot \begin{pmatrix}
\frac{k}{n-1} & \frac{1-k}{n-1} & \cdots & \frac{1-k}{n-1} \\
\frac{1-k}{n-1} & k & \frac{1-k}{n-1} & \cdots \\
\vdots & & \ddots & \cdot \\
\frac{1-k}{n-1} & \cdots & k & \frac{1-k}{n-1}
\end{pmatrix} \\
&= \left(\frac{k}{n} + \sum_{j=1, j\neq i}^{n} \frac{1}{n} \cdot \frac{1-k}{n-1}, \ldots\right) \\
&= \left(\frac{k}{n} + (n-1) \cdot \frac{1-k}{n \cdot (n-1)}, \ldots\right) \\
&= \left(\frac{k}{n} + \frac{1-k}{n}, \ldots\right) \\
&= \left(\frac{1}{n}, \ldots\right) = \mathbf{u}
\end{align*}
\]

\[\square\]

The convergence speed of \(\mathbf{P}_k\) is controlled by \(k \in \left[\frac{1}{n}, 1\right)\). A bigger \(k\) results in a faster convergence to the 'no knowledge' belief state by applying \(\mathbf{P}_k: \lim_{t \to \infty} \mathbf{P}_k^t = \mathbf{u}\).

To compare the convergence for different values of \(k\), the distance of a belief state \(\mathbf{b}\) to \(\mathbf{u}\) is defined by \(\Delta(\mathbf{b})\), where \(\| \cdot \|_2\) represents the Euclidean Norm:

\[
\Delta(\mathbf{b}) = \left(\|\mathbf{b} - \mathbf{u}\|_2\right)^2 = \sum_{i=1}^{n} \left(b_i - \frac{1}{n}\right)^2
\]

(4.5)

**Proposition 5.** For \(\frac{1}{n} \leq q < k < 1\), and \(n \in \mathbb{N}, n > 1\) the distance \(\Delta(\cdot)\) of the result to the uniform belief state of propagating an arbitrary probability vector \(\mathbf{b} \neq \mathbf{u}\) through \(\mathbf{P}_q\) is smaller than propagating it through \(\mathbf{P}_k\). It holds: \(\Delta(\mathbf{P}_q\mathbf{b}) > \Delta(\mathbf{P}_k\mathbf{b})\) iff \(k > q\) for an arbitrary probability vector \(\mathbf{b}\).
4.4 Modeling oblivion

Proof.

This proof is only valid for \( n > 1 \) and \( \mathbf{b} \neq \mathbf{u} \), which is intuitively clear since \( \mathbf{u} \) is the fix point of \( \mathbf{P}_k \) independently from \( k \).

It is intuitively clear that \( \Delta(\mathbf{P}_k \mathbf{u}) = \Delta(\mathbf{P}_q \mathbf{u}) \) holds true, since \( \mathbf{u} \) is the stationary distribution of \( \mathbf{P} \).

Tracking a one-dimensional belief space \( (n = 1) \) is trivial: In the case of a one-dimensional belief space \( n = 1 \), the only valid matrix is \( \mathbf{P}_k = (1) \) which is the identity matrix in one dimension. Even this matrix satisfies the convergence and fix-point criteria, since the only valid 1-dimensional belief vector is the vector (1), which is the fix-point, too. However, a 1-dimensional belief vector estimating one system state is, in principle, meaningless.

\( \mathbf{P}_k \) represents the process of oblivion as matrix. It drives an arbitrary belief state \( \mathbf{b} \) stepwise to the uniform belief state \( \mathbf{u} \). The parameter \( k \) controls the level of convergence, which can be intuitively interpreted as the speed of forgetting. If \( k = \frac{1}{n} \), the process of oblivion leads within one step to the uniform belief state. A large \( k \) results in greater distance given by \( \Delta(\cdot) \) and a lower convergence speed.

Figure 4.7 shows the convergence of the oblivion process for different \( k \) over a three-dimensional state space starting at the belief \((1.0, 0.0, 0.0)\), which represents the system to be in the first of three states with absolute certainty. The euclidean distance to the target, the uniform belief state, is plotted. Converging against zero confirms that the belief state converges towards the uniform belief state \( \mathbf{u} \).
4 Controller design

Figure 4.7: Convergence of the oblivion process

4.5 Summary

This chapter introduces the characteristics of problems that can be expressed and solved by POMDPs. It is stated that simpler methods might solve the given problem more efficiently if one of the problem aspects (see Section 4.1.1) can be relaxed.

Four different processes of modeling POMDPs are introduced. Based on the dependencies of the model elements (states, actions, observations, observation model, transition model and reward model) manual modeling and automated model deduction is introduced and discussed in detail.

The novel paradigm of an estimation controller is presented. It extends common estimation with a cost function that represents the objectives of changing the estimate and the estimation error. This allows to develop an estimated based on a strategic control policy with respect to multiple objectives.

Finally, a novel process model for modeling oblivion is proposed. It enables to deal with event observations. If no observation takes place, it develops the estimators internal belief state step-wise towards no knowledge. Technically, it converges the discrete belief state of a Bayesian estimator from an arbitrary probability distribution towards the uniform belief state, which represents no knowledge about the system state. It is mathematically proven, that the process uniquely converges and it is shown that the convergence speed can be controlled by a scalar parameter.

The paradigms of the oblivion process and the controlled estimator are applied to the parallax error correction, which is presented in Chapter 5.
Case study: Parallax error correction on interactive screens

In Chapter 1 intelligent agents were motivated in general. Chapter 2 presented the methods of Bayesian estimation, MDPs, HMMs and POMDP in detail. Chapter 3 showed the current state of the art of applying MDPs and POMDPs to real world problems and Chapter 4 presents the modeling process. The necessary problem characteristics are listed, simplifications are shown and variants of modeling problems as POMDP are presented. The theoretical bases of an estimation controller and an oblivion process for Bayesian tracker are presented. In this chapter, the theory is applied to a real world problem: Reducing the parallax error on interactive screen.

This chapter describes the research on correcting the parallax error on interactive screens which occurs due to an offset between interaction plane and image plane. As it will be shown, the parallax error impairs the touch precision. To increase the interaction precision reducing the parallax error without the need of additional hardware, an interactive intelligent system is implemented. An interactive intelligent system is an intelligent controller that the human interacts with [112]. It combines the research fields intelligent systems and human-computer interaction.

Outline

Section 5.2 introduces the field of human-computer interaction, in detail interactive systems. It defines the interaction space and widget-based graphical user interfaces in terms of hardware and software components. In Section 5.3, the problem of the parallax error is established and potential approaches to overcome the problem and increase the interaction precision are introduced.

The related work (Section 5.4) shows the state of the art in terms of increasing the interaction accuracy on interactive screens. Common calibration techniques of interactive screens are summarized and is listed how high precision of touch interaction is achieved or it’s need avoided so far. It is shown how spacial displacement of haptic feedback, which is caused by the parallax error, irritates the user, and how interaction quality is evaluated. The section concludes with the research gap of correcting the parallax error on interactive screens.

To overcome the parallax error by estimating the user’s viewpoint with a Bayesian filter, several models describe the nature of the human (the system) and observations in terms of interactions on the screen. To set up the models properly, the system - the human user behavior - is identified in Section 5.5. The models are deduced from measuring the user with high quality tracking devices in a lab environment, while using an interactive system.

In the following section, correction controllers are presented. A tracking controller which corrects the
5 Parallax error correction

parallax error by directly tracking the users viewpoint is developed in Section 5.6. The POMDP controller is developed in Section 5.7. It translates the given problem formally into a sequential decision problem. The POMDP model topology is defined in detail before the model is deduced from the measurement data (see Section 5.5). In showing the controller architecture it is described how to integrate the controller in a common HCI interface and how to deduce the control policy by planning.

Finally, Section 5.8 describes the controller evaluation. It analyses the correction behavior of the POMDP controller for typical scenarios. The chapter closes with a conclusion, that critically evaluated the usage of POMDP controller for the problem of correcting the parallax error on interactive screens.

5.1 Artificial intelligence in human-computer interaction

An intelligent controller implements capabilities that have traditionally been associated more strongly with humans than with computers, such as the abilities to perceive, interpret, learn, reason, plan and decide. Systems that exhibit such capabilities mostly use techniques that are originated in the field of Artificial intelligence (AI). Reasoning and planning enable an interactive system to anticipate the user’s future behavior, while learning makes the system to adapt to the user. Hence, intelligent interactive systems extend common Human-computer interaction by methods from AI.

Human-computer interaction (HCI) considers the interaction between humans and machines, mainly in terms of psychological aspects (capabilities of the human) and technological systems (capabilities of the machine). The last years have shown vast progress in the development of machine hardware interfaces. Today, interactive displays, a composition of a digital display and touch sensitive surface, are common even for small electronic devices like smart phones. Nowadays, the development of the technical systems tends to be satisfied. Today, the hardware for interactive systems is a mass-product.

The HCI research community moves from focusing on technological aspects of devices and applications to the human aspects. As an example, the topics of the ACM Symposium on User Interface Software and Technology (UIST) in the beginning of the 21st century focused on interaction hardware in terms of pointing, touch sensing, projection and mobile interfaces. Today, the focus has moved to more complex interfaces like augmented reality and interfaces for blind people and additional topics like social aspects and assistant systems which focus on the human side are considered.

Adaptive software applications reduce explicit human action utilizing new input modalities, for example the geographic location, orientation [159] or the user’s position or gestures, which takes place at the continuous space in front of the computer display. To automatically adapt to the human, the computer needs to capture it. Therefore, the computer interfaces are extended with cameras and 3D tracking devices capturing the user, GPS sensors capturing the geographical location and accelerometers, compasses as well as gravity sensors gathering orientation and movement of the device. Since measurements, in particular measurement of cheap sensors, do fail and are influenced by noise, signal filtering is applied to establish reliable data. To additionally interpret natural interaction like human gestures and to anticipate the human behavior, methods from AI are used in this field of application.

Common HCI research focuses on improving the interaction design of human computer interfaces based on the current applications by studying the involved users. AI, on the other hand, focuses on improving the insides of the controller - the algorithms of applications (or intelligent machines) and its technical evaluation, such as computation time and memory usage. Due to Grudin [94], HCI originated in psychology, AI in mathematics and engineering. He also states that the two research communities HCI and AI did commute notably but still sporadic, although AI technologies seek for input from the application side - in this particular case the HCI community.
5.2 Human-computer interaction on interactive screens

The field of intelligent interactive system combines research on HCI and AI. HCI is the application domain that formulates the problem, while AI represents the solution domain providing methods to solve the given problems. From the AI perspective, HCI is a challenging field of application, since interaction with humans is complex (i.e. natural language processing). From the HCI point of view, AI provides a toolbox to solve problems. Hence, the interface between both domains is the problem formulation in terms of models.

By combining the two fields, the technical development of AI gets influenced by the user behavior (i.e. search heuristics) and studying the user is now affected by the advanced decisions of the intelligent system. In particular, model-based controller systems utilize a model of the agent’s environment. Modeling the world, or, in case of HCI, the user allows exploiting knowledge about the human behavior from the machines perspective. To anticipate the future from reasoning and act in a non-myopic way, the environment’s system dynamics are modeled and planned out automatically. Measurement models express the effect of sensor errors and make the controller act carefully considering uncertainty in developing strategic control policies. To benefit from model-based controllers in the field of intelligent interactive systems, the agent’s modeled environment is the human, which is uncommon in the field of applied AI.

Hence, applying AI to human-computer interaction, in particular interactive screens which will be introduced in the next section, is an interesting research field.

5.2 Human-computer interaction on interactive screens

Technically, Human computer interaction addresses the field of interaction between humans and computers (see Figure 5.1). As schematically shown in Figure 5.1(b), the interaction involves two channels: the information flow from the user to the machine and vice versa.

In general, the human user perceives information visually through his eyes, auditive through his ears and tactualy through sensors in muscles and skin. Vision is often the dominant sense in real environments. For example, when a subject is moving his hand along a planar surface while wearing glasses, which distort the visual channel, the user assumes to feel that the surface is curved [91]. Vision also domi-
5 Parallax error correction

...nates proprioception in virtual environments, which was found by Burns et al. [57]. Due to the visual dominance, the human-computer interface focuses on visualizing content on digital screens (monitors, displays).

In this section, the human-computer interaction is introduced focusing on interactive screens. The main aspect of interactive screen is the interaction space in order bridge the gap between humans and computers. The underlying concept is introduced in the following section. Then, the software-side of the graphical user interface, in particular widget-based interfaces, and the hardware, in terms of display and tracking technology, are introduced.

5.2.1 Interaction space

The user potentially acts by hand or voice on digital content. Due to the high complexity of automatic natural language processing, haptic touch-based input is the commonly preferred information channel from human to machine. Usually, commands to the machine are expressed using separate input devices like keyboards, pointing devices (i.e. computer mouse, trackball, touchpad) or touch panel. However, these common input devices spatially separate the input and output information channel within the loop of information flow between humans and computer applications, as schematically shown in Figure 5.2(b). Today, a desktop workstation or mobile laptops are set up as horizontally oriented keyboard and mouse (input) which are located in front of the vertically oriented display. In contrast, interactive screens spatially combine the input and output devices by mounting a touch sensitive surface in front of the display; see Figure 5.2(a).

Examples are mobile phones and electronic tablets, ticketing machines, horizontally mounted tabletops [125] and vertically oriented, large scale interactive whiteboards [127]. Due to the orientation, tabletops are highly stressed in terms of physical load and equipped with a robust interaction plane.

Manipulation of digital objects is either done by the user’s bare finger or manipulating by tangible user interfaces (TUIs) [111] like pens or erasers [105] [106]. Indicating a specific function, which is known to the user from daily use, handling TUIs is intuitive and easy to learn. Moreover, the interaction is enriched by multi touch (gestures) detection and TUI’s [106]. In the following, touch interaction express the interaction on touch sensitive surfaces no matter if executed by bare finger or TUI.

Definition 5. An Interactive Screen (or surface, display) is a device for human-computer-interaction that enables the user to directly interact with graphical content by touch interaction on the digital objects (on-screen interaction).

Interactive Screens consist of an electronic display and a touch sensitive surface. The digital monitor presents information to the user on the image plane. Mounted on top, a touch panel detects touch actions from the user to interact with the content on the interaction plane. The real interaction is mapped to the digital content by transferring the touch information onto a virtual cursor (pointer) on the graphical user interface.

Due to the ability to directly manipulate digital content on the touch-sensitive screen by finger or TUI, interactive screens have a very strong appeal to users’ capturing man’s natural pointing instincts [...] as mode of HCI communication’ [161]. In particular, novices benefit most from the intuitive interaction: Touch screens provide a highly intuitive usable and easy-to-learn human computer interface. A fast learning curve makes touch screens an ideal interface for interacting with public installations. Hence, interactive surfaces become more and more state-of-the-art as human computer interaction technology replacing spatially separated input devices, i.e. mouse and keyboard.
5.2 Human-computer interaction on interactive screens

The close coupling between input and output, control and feedback, hand action and graphical user interface, real world and virtual world on interactive screens enables an instantaneous manipulation. On interactive screens, the user manipulates digital objects with touch actions instead of using a spacially separated input device, i.e. mouse or keyboard. Besides its directness, touch screen interfaces have the following advantages. First, no extra input control device or space is necessary for touch screen interaction, since the control surface is overlaid on the display. Second, touch screens are more robust and intuitive to use than separate input devices, which is useful in cars and public usage, i.e. ticketing machines and ATMs.

Large vertically oriented interactive screens are used as electronic whiteboards (e.g. SMART® Board [20]); and horizontally oriented tabletops [124] are already widely spread for distributed collaborative work, as presented by Brave et al. [55] and Ganser et al. [89]. Due to the size, large interactive screens run multiple applications a time and allow multi-user interaction, i.e. locally sharing the workspace for team work [126, 88]. Digital screens allow, moreover, sharing the workspace remotely [127], which allows working in spacially distributed teams. Today’s personal mobile phones and electronic tablets are commonly equipped with interactive screens instead of keyboard and pointing device. Touch screens are also widely deployed in kiosk applications in public area, e.g. ticketing machines and cash machines, as shown in Figure 5.4. In contrast to personal devices, applications in public area deal with frequently changing users.

To assure high quality interaction, precise interaction is necessary. It relies on a good spacial alignment between the touch position on the interaction plane and the position of the aimed digital object, which is shown on the display. Hence, an exact calibration of the touch detection system is necessary for an exact user interaction.

However, to support physical robustness a protective glass layer is attached in a certain distance to the image plane. The resulting offset between the interaction and image plane causes a parallax error that reduces the interaction precision of the interactive system and leads to interaction errors and unintended behavior of the computer. The parallax error stems from the offset between interaction and image plane and depends on the user’s changing viewpoint. The user’s viewpoint is biased by his physique (height and arm length), and depends on the position of the currently aimed button element on the screen.

By directly capturing the user’s viewpoint, the parallax effect onto the interaction accuracy could be eliminated. Such a tracking based controller extends the interaction space from the two-dimensional interactive plane of a touch display to the three-dimensional space in front of the screen.
5 Parallax error correction

Continuous interaction space

Extending the interaction space from the 2D interaction plane to the 3D space in front of the interactive screens is introduced as Continuous interaction space by Marquardt et al. [142]. They track the user’s hand above a tabletop and propose gesture patterns to manipulate the digital content on the screen. Extending display interaction by the z-axis is called Smart Interactive Displays by the Applied Sciences Group of Microsoft Research in 2011. As shown in Figure 5.3(a) the continuous interaction space combines the two modalities touch interaction, which takes place directly on the interactive surface, and hand gestures above the surface. Tracking any user characteristic (i.e. view point position) in front of the screen extends this definition [187].

An example application is the automatic content orientation on interactive tables as proposed by Schlatter, Migge and Kunz [187]. On horizontally aligned digital tables multiple users are located around the digital surface. In order to be readable, the orientation of the content depends on the viewpoint of the user. By tracking the users’ viewpoint, the digital sticky note is oriented automatically as shown in Figure 5.3(b).

5.2.2 Widget-based graphical user interface

2-dimensional graphical user interfaces (GUIs) are an instance of human computer interfaces that show images instead of plain text to the user. GUIs show information in terms of digital objects as icons. The GUI provides functions on the digital objects, i.e. selection, copy, open, move and remove. GUIs enable drawing and sketching applications, like Computer aided design applications, as well as virtual reality environments. GUIs show virtual objects in the display coordinate system.

Definition 6. The display coordinate system is defined as three-dimensional left-handed coordinate system. On vertically mounted displays, the x-axis is aligned horizontally; on horizontal oriented screens, the x-axis is aligned horizontally to the common orientation of the digital content. The y-axis is oriented orthogonally to the x-axis within the image plane. The z-axis thus is always perpendicular to the screen. The origin is located on the upper left corner of the display. The display coordinates are measured in pixel (px).
5.2 Human-computer interaction on interactive screens

Graphical representation of virtual objects

Common 2-dimensional graphical elements are buttons, bars, text boxes, icons and images as well as menus, windows and layouts. Due to the wide range of different functionalities, the interactive graphical components are called widgets. Although widgets provide the same basic functionality, they can be divided into groups with respect to their purpose: Interactive elements represent a certain function, which is triggered by interacting, for instance ‘clicking’ it. Information widgets, like text boxes or images, mainly provide information to the user without interaction. Layout widgets are abstract widgets. They do not represent a digital object by itself and are used to orientate and order child widgets. Windows are commonly used in the group of layout widgets and separate widgets from other to indicate a specific, e.g. application, context in the window manager. They often provide additional icons for closing, minimizing or hiding the window.

The GUI is controlled by the window manager (windowing systems). It shows digital content on the screen and processes the user’s input from pointing devices. The graphical representation is implemented in a hierarchical structure starting at the root window of the window manager. Each user application registers at least one window at the window manager. If the focus is set to the application window, the interaction events from the input devices (see next section) are forwarded from the windowing system to the application. In the context of object oriented programming, events represent a certain occurrence, which is exchanged between objects as a message. The message describes the event in detail with additional parameters.

Starting at the application window, the widgets are organized as tree: Each widget has a parent that defines the graphical orientation in terms of position and layer, and the event chain. After getting the interaction event from the window manager, the application sends the event to the topmost widget at the position of the occurrence of the event. If the widget refuses to accept the event, it is forwarded to its parent up to the root window.

For technical details see Appendix B.2.1.

5.2.3 Input hardware

To manipulate digital objects on the screen, keyboard and pointing devices are used. The pointing device controls a virtual pointer on the screen in terms of position and command to manipulate graphical objects as described in section 5.2.2. Pointer hardware devices bridge the gap between virtual world and real world. The user’s input is sent from the input device hardware to the windowing system, which represents the information as virtual pointer and sends the actual pointer status as event to the currently active application. The pointer event information contains the position and the status, which commonly expresses the functions click, double click, drag, right click and release.

Hardware devices like computer mouse, trackballs and trackpads implement these functions by tracking movement and state, i.e pressing and releasing buttons on the device. Buxton presented a descriptive 3-state model for graphical input devices in [58]. To control the pointer position, a mouse is moved on a flat, spacial separated surface, a trackball measures the rotation of a ball and a trackpad detects the movement of the user’s finger on a small input panel. Since the input area of the input device and the controlled area of the virtual point on the screen are not collocated, the pointer position is controlled by the relative movement of the hardware instead of its absolute position. Hence, the calibration only needs to consider mapping movement, speed and velocity.

Interactive screens support a direct manipulation of digital objects at the screen, tracking the users input
5 Parallax error correction

directly on the screen. The collocation of the digital content (virtual objects) and the user’s input (real word objects) enables a highly intuitive manipulation with direct control of graphical objects. Contrary to spacially separated pointer inputs, on interactive screens the virtual pointer is controlled by the absolute position of the touch interaction, not by it’s relative movement. Hence, a precise calibration of the touch panel and the display panel is crucial, i.e. rotation, translation, scaling.

Display and interaction tracking technologies

Interactive systems consist of two main components: the display and the touch detection. Displaying systems show digital content to the user in front of a screen: Back projection systems show the reverse image of a projector, which is located behind the image plane. Whereas for front projection systems the user and the projector are on the same side of the image plane. Since the user’s shadow effects the digital content on the screen, back projection systems are preferred. Although projection systems provide an inexpensive, large-scale interactive system, the main drawbacks are the size of the system due to the needed space between projection plane and beamer and the sensitivity to daylight. Flat Liquid Crystal Displays (LCDs) provide a compact design using a liquid crystal matrix to electronically select the amount of passing back light emitted by light-emitting diodes (LEDs) or fluorescent lamps. Plasma displays emit light by electrically ionized small gas cells. Using flat displays instead of projection enables legroom on horizontally mounted interactive tables and wall mounted digital whiteboards.

The second component, the tracking system, is mounted on top of the display system to track the user’s interaction input: Resistive foils electrically detect touch with orthogonal conducting electrodes which are separated by a spacer. If the user touches the tracking system at a certain point, a pair of vertical and horizontal electrodes is connected mechanically and indicates the position of the interaction. Inductive and capacitive systems are based on the same principle, but measure the capacity or induction of an object, that gets close to the electrodes.

The hardware setup of interactive systems causes an offset between interaction plane and image plane. To support robustness against violence and weather conditions, the display of kiosk terminals (e.g cash machines). A distance of centimeters is common as shown in Figure 5.4. The same holds true for large interactive screens to withstand mechanical loads in particular on tabletop systems, where the user supports oneself on the table. Additionally, the projection layer is mounted with a certain distance on top of the display system to avoid Newton rings.

Except for the front projection system, which is barely used today, each interaction technology has an inherent offset between interaction and image plane: On rear projection systems, the interaction is detected at the front side of the interaction plane, while the image is projected at the screen’s backside. The conductors of the resistive, capacitive and inductive systems are mapped on top of the image plane with a certain diameter, due to physical robustness.

The following section introduces the parallax error formally before the currently applied solutions of correcting or avoiding the parallax error are described in detail in Section 5.4.

5.3 Problem definition

A precise interaction is crucial for many touch screen applications to detect interaction with small targets and enable exact drawing. For instance, touch screen keyboards visualize very small keys to not overlay the application content with the large number of simultaneously shown keys. The targets on small screens
5.3 Problem definition

Figure 5.4: Offset between interaction and image plane

of mobile devices (phones and tablets) are small if the shown content is similar as it is usually shown on larger devices, i.e. websites.

On interactive surfaces, the user interacts directly with the virtual objects. As on real tables and whiteboards, the user expects the location of the digital targets on the interaction plane. However, digital objects are shown on the image plane, which is not necessarily collocated with the interaction plane, as illustrated in Figure 5.4 and 5.5. Hence, Interaction position on the interaction plane and Focal Position on the image plane differ with respect to the display coordinate system.

**Definition 7.** The Interaction Position is defined as position of the user’s interaction with the system. Since it takes place at the interaction plane, it is given in 3-dimensional display coordinates. Two dimensions are aligned to the display plane and the third is aligned orthogonally to the display orientation.

The interaction position on the interaction plane is mapped onto the detected interaction position on the image plane. The mapping is defined by the static calibration of interaction tracking system and display system. The mapping is assumed to be orthogonally to the image plane. The detected interaction position controls the position of the virtual pointer on the graphical user interface software. The virtual pointer sends interaction events to virtual objects (targets), which are shown on the display.

**Definition 8.** Targets on graphical user interfaces are interactive objects that the user can manipulate. Usually, 2-dimensional targets are areas known as widgets, e.g. buttons, icons (graphics) or text phrases used as hyper references in HTML to link web content. 3-dimensional objects are projected on the image plane.

In the following, the focus is on 2-dimensional area targets instead of 2.5 dimensional visualizations of 3 dimensional objects.
Definition 9. The **Focal Point** (or Target Point, Focal Position or Target Position) is defined as the point of the area target shown at the display which the user aims to interact with.

Definition 10. The **Interaction Error** (or pointing error) on interactive screens is defined as offset between the interaction point and the focal point. It is measured in the Euclidean Norm with respect to the display coordinate system.

Besides the system inherent distortions (the accuracy of the interaction tracking system, the alignment of the interaction tracking system and the display system), the interaction precision is also influenced by the user, in terms of: The user’s motivation to hit precisely: The user’s decision to aim at a certain point of a target on the screen. And the user’s motor ability to execute the interaction: The ability of the user to hit the aimed position. And the parallax error: The offset between the detected pointing coordinates on the interaction plane and the aimed coordinates on the image plane. (Further influences are possible.)

The user notices the error only if an interaction with a digital target does not occur as expected. Common interaction on GUI widgets are **click** and **release**. The GUI detects an interaction on a widget by detecting the interaction position within the visible area of the widget. If the interaction is not precise and unintentionally detected outside the widget, the application does not respond as expected. As a result, the user might get annoyed, since he must withdraw the misinterpreted action and redo the intended action.

Interacting with 2-dimensional targets on digital screens is affected by the limited human depth perception and leads to interaction errors. **Depth perception** is the visual ability to perceive the environment in three dimensions and the distance to objects. Humans perceive depth information in various ways: accommodation, familiar size of known objects, motion parallax, lighting, shading and convergence. The human perception is limited for virtual objects shown on digital screens. No common size, lighting
or shading can be used to perceive the object’s depth, since 2-dimensional objects are artificially and
defined per pixel (not in real world metric, i.e. millimeter!) and its visualization on the screen depends
on the varying size of the screen. Under real world viewing conditions, accommodation and convergence
eye movements vary correspondingly and dependent on object distance. Watching virtual objects on a
digital screen, the eyes must maintain accommodation to the screen’s distance [214]. Hence, the depth
perception is limited for targets on digital screens.

Hence, it is reasonable that the user interacts on the direct line between his viewpoint and the virtual target
to hit the target correctly utilizing the overlap between his finger and the target. Due to the offset between
interaction and image plane, the user will hit the interaction plane on the line between his viewpoint and
the target. Since the interaction position is commonly mapped orthogonally onto the image plane, the
detected interaction position is shifted and the interaction does not hit the target correctly, as shown in
Figure 5.6.

Due to the Oxford English Dictionary, the Parallax is the difference in the apparent position of an object
as seen from two different points [156]. The parallax is defined as ‘the difference in direction of a
celestial object as seen by an observer from two widely separated points’ [80]. It defines ‘the apparent
displacement of an observed object due to a change in the position of the observer’ with respect to its
background. The parallax causes a displacement in the apparent position of an object viewed along
different lines of sight. Hence, the relative position of an object in front of another object changes,
depending on the viewpoint. In the context of interactive systems, the observed object is the aimed target
and the observer is the user. The displacement affects the interaction point between the virtual target and
the user in front of the screen.

Definition 11. The Parallax Error on Interactive Screens is the displacement between the virtual po-
sition of the interaction from the touch panel (interaction plane) and the corresponding target on the
image plane. It stems from the parallax effect for a given object on the display, assumed that the user
interacts with a digital object on interactive screens in the line of sight.

It affects the interaction position on the touch panel for a target on the display, due to the offset be-
tween interaction plane and the image plane. The resulting interaction error affects the accuracy of any
interaction on touch sensitive surfaces.

Figure 5.6 shows the effect of the parallax on interactive screens for two users with viewpoint A and B.
Aiming to hit target point 5, User A interacts at point 3, which is in the line of sight between Viewpoint A
and target 5. User B, however, interacts at a different position 1. If assumed that the interaction position
on the interaction plane is mapped orthogonally onto the image plane, the resulting interaction error for
user B is $E_x$. This so-called parallax distortion can be derived from the geometry. It is for Viewpoint B:

- $E_z$: Offset between image plane and interaction plane
- $V_x(V_z)$: Distance from Viewpoint B to the target point 5 in x (z) direction

$E_x$ is the parallax distortion in x direction between the detected interaction point 6 and the target point
5. Figure 5.6 also shows the different interaction points 1 and 3 for different viewpoints A (B) aiming
for the same target point 5. Due to the offset $E_z$, the interaction is detected at point 4 (6) instead at point
5, which would be the only correctly interpreted interaction position, if the interaction tracking and the
display system are orthogonally aligned. Only interaction point 2 would cause a precise interaction for
the target at point 5.

The resulting horizontal error $E_x$ (similarly for vertical $E_y$) for viewpoint B can be easily calculated by:
5 Parallax error correction

![Diagram of Interaction Plane and Image Plane]

**Figure 5.6: Geometrical dependence of the interaction point on the viewpoint**

\[
E_x = \frac{E_z \cdot V_x}{V_z + E_z}
\]  

(5.1)

Note that this equation only takes the geometric conditions into account and omits the refraction indexes on the different media. The difference in terms of attenuation is negligible for the transition between the media air and glas.

The parallax error changes according to a changing viewpoint, which occurs on turn taking of multiple users or when the user moves to reach a certain target on a large interactive screen.

Example: Assumed the interaction system to be orthogonally calibrated, which means that every interaction point is mapped orthogonally onto the underlying display position. For an offset of 15 mm between the interaction and the display plane \((E_z)\), the parallax error is already 9 mm for a viewpoint offset of 30 cm \((V_x)\) and a screen distance of 50 cm \((V_z)\), which is realistic as the measurement results in Section 5.5.1 show. As a result, the user would not hit a target widget (e.g. button) of the size up to 17 x 17 mm without any further error correction.

The parallax error is subject to the following hypotheses.

**Hypothesis 1.** The parallax error on digital screens exists and it depends on the user’s viewpoint.

**Hypothesis 2.** The parallax error can not be corrected by static calibration, since the user’s viewpoint changes over time due to the user’s movement in front of the screen, turn taking between different users which might differ in height in case of multi-user interaction or systems that are used in public area.

**Hypothesis 3.** Left- and right-handed user are distinguishable by their interaction error, since the parallax error depends on primary hand. Left-handed people are located right of the target and interact right of the target (parallax error) and right-handed people interact left of the target, due to the relation between hand, interaction error and viewpoint.

5.3.1 Parallax effect measures

Based on the hypotheses in Section 5.3, the first question to be answered is: **Is the parallax error measurable on statically calibrated interactive screens with an offset between interaction plane and image plane?** This question will be answered by proving that the interaction error correlates with the user’s viewpoint. Showing that the user acts in line of sight on interactive screens proves that the parallax error, as part of the overall interaction error, stems from the user’s viewpoint.
5.3 Problem definition

In order to demonstrate the parallax error, a user study is conducted, which monitors the user interacting on a large screen (DigiBench) with a significant offset between interaction plane and image plane of approximately 10 mm between interaction and image plane. The user study is described in detail in Appendix A.1.

Results

Based on the 3D measurement of the touch position on the interaction plane, the target position on the image plane and the user’s viewpoint position in front of the screen, the following data is calculated. First, the interaction error (see Definition 10) is deduced as offset between the center of the target and the actual interaction position on the screen. Second, the assumed interaction position is deduced from the target and the viewpoint location according to Equation 5.2. And the dexterity is given as deviation of the interaction position from the assumed interaction position, in line of sight. It indicates the influence of the parallax error onto the overall interaction error, since it represents the error of the optimally parallax corrected interaction. The dexterity is assumed to be lower than the interaction error.

Two aspects are analyzed: The pointing accuracy is a measure of how well the user hits the given target. It is indicated by the mean value of the measurements. The pointing precision, on the other hand, refers to the stability (variance) of the data and is a measure of the quality of repeatability. Figure 5.7 shows the vertical and horizontal click error with regard to the center of the target. Interaction error above 60 mm are treated as unconsidered outliers.

The measurements are summarized in Table 5.1. The horizontal measures show that the user did mostly click left of the target’s center with a mean distance of -5 mm. The histogram of the horizontal interaction error shows two clusters around -5 and 0 mm. The mean of -6 mm in vertical error indicates that the users mostly clicked above the target. With 4 mm, the standard deviation is similar to the horizontal values. The horizontal dexterity has a mean value of -3 and a standard deviation of 2.55 mm, which is lower than the interaction error (mean -5, \( \sigma = 4.19 \)). This does not hold true for the vertical dimension (see Table 5.1). Due to the unimodal and almost symmetric shape of the discrete distributions, the dexterity error is significantly distinguishable from the interaction error, due to the non-overlapping center (50%) quantiles. Although this does not hold true for the vertical dimension, the mean of the vertical dexterity (-5 mm) is lower than for the interaction error (-6 mm).
### 5 Parallax error correction

<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>1st quantile</th>
<th>Mean (median)</th>
<th>3rd quantile</th>
<th>Max</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Horizontal</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interaction error</td>
<td>-14.58</td>
<td>-7.72</td>
<td>-5.13 (-6.86)</td>
<td>-2.57</td>
<td>7.72</td>
<td>4.19</td>
</tr>
<tr>
<td>Dexterity</td>
<td>-8.70</td>
<td>-4.92</td>
<td>-3.18 (-3.00)</td>
<td>-1.49</td>
<td>5.32</td>
<td>2.55</td>
</tr>
<tr>
<td><strong>Vertical</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interaction error</td>
<td>-14.54</td>
<td>-8.88</td>
<td>-6.75 (-8.08)</td>
<td>-4.84</td>
<td>7.27</td>
<td>3.95</td>
</tr>
<tr>
<td>Dexterity</td>
<td>-15.28</td>
<td>-8.51</td>
<td>-5.63 (-6.14)</td>
<td>-3.72</td>
<td>12.48</td>
<td>5.03</td>
</tr>
</tbody>
</table>

*Table 5.1: Interaction error and dexterity characteristics [mm]*

<table>
<thead>
<tr>
<th></th>
<th>Error left of target</th>
<th>Target hit</th>
<th>Error right of target</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Horizontal</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No correction</td>
<td>52.3</td>
<td>46.5</td>
<td>1.1</td>
</tr>
<tr>
<td>Optimal correction</td>
<td>9.3</td>
<td>90.6</td>
<td>0.0</td>
</tr>
<tr>
<td><strong>Vertical</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No correction</td>
<td>61.6</td>
<td>37.2</td>
<td>1.1</td>
</tr>
<tr>
<td>Optimal correction</td>
<td>53.4</td>
<td>43.0</td>
<td>3.4</td>
</tr>
</tbody>
</table>

*Table 5.2: Hit rate with and without correction [%]*

Without any correction, the subjects hit the given target in 46.5% horizontally and in 37.2% vertically, as Table 5.2 shows. Since 41% (45 %) of the horizontal (vertical) clicks lie outside the target area, 19 % of the interactions with the target area are recognized correctly. Overall 82% of the interactions lie outside of the target area (13 × 12 mm).

### Conclusion

It is shown by example that the interaction error is affected by the parallax error, as stated in Hypothesis 1. It is moreover exposed that the parallax error in principle depends geometrically on the viewpoint, as shown in Figure 5.6.

Moreover, studying the effect of the parallax error confirms Hypothesis 1. The parallax error stems from the user’s viewpoint.

Under the assumption of a perfect parallax error correction (shifting the interaction position to the assumed interaction position), the dexterity gives information on the remaining interaction error. Perfect parallax error correction would have increased the hit rate from only 17% to 38%, which is due to the strongly increased horizontal hit rate from 46.5% to 90.6%. The vertical error, however, does differ. Hence, the parallax error might not be optimally corrected with a ‘bang-bang’ strategy rather than by a step-wise correction adaption.

### 5.3.2 Potential improvement of interaction precision

The quality of an interaction system can be quantified by counting the number of successfully recognized interactions. A correct interaction is defined as hit detection of the target, which occurs if the pointing coordinate is located within the target area. An imaginary perfect parallax correction controller directly measures the viewpoint of the user with a 3D tracking system. It shifts the interaction position by the
offset between assumed interaction point and target center in order to eliminate the parallax error. The assumed interaction position is defined on the line between viewpoint and target (see Figure 5.6), which implies not to be affected by the parallax error. Although this controller could eliminate the parallax error, it does not reduce the error that stems from the motoric skills and the motivation where to hit the area target.

The deviation between the resulting target hits indicates the potential improvement of the parallax correction with respect to the specific target (shape and size) and the interaction system. Section 5.3.1 shows that applying the imaginary perfect parallax correction controller would double the number of correctly detected pointing interactions from 17% to 38% (up to a hit rate of 90.6% in the horizontal and 43% in the vertical dimension) at the large interaction screen test setup DigiBench.

5.3.3 Correction bounds

As described above, the parallax error depends on the viewpoint of the user. In this section, the correction is estimated formally to show the theoretical potential of correcting the error.

The parallax error $e_{in}$ is given by the offset between the target and the mapping of the interaction point $I$ onto the image plane (detected interaction position), which is assumed to take place orthogonally. As already shown for one dimension in Equation 5.1, $I_{x,y} \in \mathbb{R}^2$ is deduced from the viewpoint $V_{x,y,z} \in \mathbb{R}^3$ and the screen offset $o$ by:

$$I_{x,y} = V_{x,y}(1 \cdot o)(V_z + o)^{-1}$$

(5.2)

Under the assumption that the correction controller shifts the detected interaction position by its inverse (negative) value, the parallax error is corrected optimally. Every deviation, which might stem from other errors like the user’s motor skills (see above), is expressed as $e_{out}$. It is the error after correcting the parallax with respect to the viewpoint, as defined in equation 5.4. The correction controller decreases the interaction error if $|e_{out}| < |e_{in}|$ holds true. To identify its bounds, equation 5.5 and 5.6 establish the lower and upper bound of decreasing the interaction error with respect to the user’s viewpoint $V_{x,y}$. Equation 5.5 shows that the interaction accuracy improves, if the viewpoint is detected on the correct side of the target (lower bound) and equation 5.6 shows its upper bound of $2 \cdot e_{in}$.

The means, if a correction controller that estimates the user’s viewpoint does detect the side of viewpoint with respect to the target correctly and the correction value is lower than two times the error without correction, it increases the interaction accuracy by reducing the parallax error. The overall 2-dimensional error is reduced component wise, as shown in equation 5.3.

$$|e_{out}^x| < |e_{in}^x| \Leftrightarrow (|e_{out}^x| < |e_{in}^x| \land |e_{out}^y| \leq |e_{in}^y|) \lor (|e_{out}^x| \leq |e_{in}^x| \land |e_{out}^y| < |e_{in}^y|)$$

(5.3)

Hence, the following equations consider $I$ and $V$ in one dimension only.

$$e_{out} = e_{in} - I$$

(5.4)
This section analytically investigates the nature of the parallax error. In the following sections it is shown that the parallax error exists on a real interactive system and how the interaction precision could be increased by correcting it.

5.3.4 Automatically correcting the parallax error

Under the assumption that the user interacts in the line of sight, the controller linearly extrapolates the intended target position from the viewpoint and the interaction position to overcome the parallax error. Although the tracking controller is potentially optimal, the hardware installation is expensive and causes privacy issues.

Instead of directly capturing the viewpoint, a continuously recalibrating controller utilizes the interaction error of the user’s touch interaction on the interactive surface to estimate the viewpoint, which is effected by uncertainty. Like the tracking controller, the estimator relies on the assumption that the user interacts in the line of sight on the target. The viewpoint is deduced from the interaction error and the target on the screen. In contrast to the tracker, this software driven approach is less expensive since no additional hardware is needed to correct the parallax error. It utilizes advanced methods from AI to compensate the highly uncertain sensor information.

The following chapter shows that actual system do not overcome the parallax error.

5.4 Related work

In this chapter, the state of the art in terms of increasing the interaction accuracy on interactive screens is presented. It shows the actual systems, which do not overcome the parallax error without major drawbacks.

First, the common way to calibrate the interaction and the display subsystem of interactive systems is summarized. Then methods to achieve and avoid the need for high precision touch interaction are
5.4 Related work

The irritation effect of displaced visual feedback assists the negative effect of the parallax error on the quality of interactive systems. The interaction quality is mostly evaluated using Fitts’ Law [85], which is introduced in detail before presenting the concept of a spacial extended interaction space. The chapter concludes with a summary and the resulting research gap.

5.4.1 High precision direct touch interaction

Direct touch pointing is the preferred interaction mode on interactive surfaces. In direct touch, the finger (or TUI device) is initially located above the touch surface. The proximate touch event is detected either immediately when the user lands on the surface, shortly after he landed, or when he lifts the finger. Hence, difficulties with precise interactions on touch screen devices are addressed by extending the processing of the touch interaction. The benefit of the second approach is that the user might correct the interaction position while he touches the surface. Potter et al. compare three touch detection techniques [168]. They found that the Take-off formalism does perform best, although it might be not intuitive to trigger a virtual object by release.

Since the user’s finger is detected as blob on the interactive surface, but the pointer position is represented as interaction point on the screen, the touch interaction must be mapped onto a single point. A reasonable approach is the center point of the blob. Lee et al. investigated bare finger direct touch input without feedback, and found the target size on touch screens to be at least the size of the finger tip respectively a width of 22 mm [131], which does not necessarily hold for using pens with a fine tip. Holz et al. studied this phenomenon of discrepancy in the location of the target point, as imagined by the user and the interaction point as measured by the touch surface [107, 108]. They found that targets have to be more than 4 mm in diameter for reliable selection. Investigating user precision in touch pointing, Berard and Rochet-Capellan found that ’the user’s precision for linear pointing is about 150 dpi or 0.17 mm’ [51], which is clearly below the size of the finger tip.

The related work on touch precision shows that the user is capable of precise touch interaction and high precision interaction is crucial on interactive systems.

5.4.2 Static display calibration

To enable an intuitive and precise interaction, the tracking and the display system must be aligned exactly. The calibration of interactive screens maps the touch position to the corresponding display coordinates. A correct mapping assures that the system detects the digital object from the user’s touch interaction, which the user intends to interact with. On the other hand, a decalibration leads to pointing errors, which occur as offset between the position of the digital target and the actually detected position by the tracking system.

The calibration of interactive screens has two aspects: First, the display system must be set up in terms of correcting optical distortions and, second, the display and the tracking system must be aligned to match the tracking position with the corresponding display position.

Aligning the projection source with the projection plane, common calibration techniques eliminate the static geometric distortions by a keystone correction. Initially developed to establish geometrical alignment between tracking and projection, for instance on cathode ray tube (CRT) screens, this field is well studied [133, 70]. The parameters of the (non)-linear optical correction is set up either manually or automatically comparing the displayed image, which is captured by a camera, with the original [133]. Lee et al. present a system that automatically calibrates the geometric relationship between the projector and
the projection plane embedding a light sensor in the target surface that reads specific patterns from the projector to determine its orientation [130].

The aspects of calibration depend on the display technology. Nowadays, interactive screens use a flat screen instead of a projector with very limited geometrical calibration capabilities. For LC-screens, linear alignment (defined as correction matrix) in terms of rotation, scaling and translation is sufficient [224], if it is assumed that the tracking system’s orientation deviates within the display plane. If not, non-linear alignments of optical distortions, i.e. barrel or pincushion effects, are additionally applied, like for common projection systems. Extending the linear transformation to a non-linearity correction enables to take the noise of the tracking sensors into account [119] and Hong-Ping et al. [222] present the correction transformation as a neuronal network.

The need for aligning the tracking and the display system stems from the fact that the tracking system is not precisely mounted on top of the display system. And even if this would be the case, the resolution of the tracking systems is typically higher than the display system, which causes at least the need for scaling calibration.

The calibration is implemented in two separate phases, the learning phase and the application phase. Since the distortions discussed so far are static, the correction parameters to align the tracking system to the display system are deduced from an initial, dedicated calibration process (learning phase) as shown in Figure 5.8(a), instructing the user to hit a finite set of targets (sampling points) on the screen. Inoue et al. describe an effective sequence of calibration points to successively refine the touch error.[110].

Since alignment can not be done automatically, the user must be involved producing interaction events to align touch panel and display (learning phase). Common correction controllers are deduced from a dedicated calibration process, where the user is asked to hit a certain sequence of pointing targets on the screen.

As shown in Figure 5.8(b), in the learning phase the system samples the pointing coordinates for a given target, which is shown to the user by a dedicated calibration application. The preprocessor transforms the pointing error from target and pointing coordinates into a correction policy, i.e. a correction matrix. By applying (linear) interpolation in between the samples, the correction matrix is globally valid on the screen for correction. In the second phase (application phase), arbitrary interaction positions (pointing
coordinates) are transformed by the correction matrix (control policy) to corrected display coordinates and forwarded to the Application. Fang et al. present an embedded calibration controller that takes into account scaling, transformation and rotation to map the tracking positions to the display coordinate system [81].

A static correction, however, does not overcome dynamic interaction errors that change over time, such as the parallax distortion. As already discussed in Section 5.3, the parallax error depends on the user’s viewpoint during the interaction. Since the user moves freely in front of the display and differs in height (e.g. wheelchair users or children), the error can not be corrected with a common static correction process. Moreover, the static calibration itself is already biased by the user’s viewpoint during the calibration process, as shown in Figure 5.9. Thus, the static calibration depends on the specific user and results in distortions for other users during collaborative work. Even for the same user, the static calibration does not take into account his changing position over time on large interactive screens. Initiating a recalibration process before each user interaction could naively solve the problem. Since this is not a practicable solution, interactive systems are only calibrated once.

5.4.3 Virtual pointer

The interaction accuracy can be increased using ‘hovering’: The system provides feedback of the current display position of a virtual cursor, which is located slightly above the interaction position on the touch panel [169]. For very small targets, however, users still tend to make a large amount of errors [35].

Another approach for precise pointing interaction is stabilizing the position of the virtual pointer by increasing the reaction time of the virtual cursor onto the touch interaction [189]. They showed that using a touchscreen was as good (in speed and pointing error) or better than using a mouse for targets sized between 13.8 x 17.9 and 6.9 x 9.0 mm. Moreover, they introduced a technique called Take-Off increasing the finger based touch interaction showing the pointer position as feedback on the screen until the aimed target is reached.

Benko et al. extend the interaction accuracy by presenting a technique that uses multi-touch interaction in order to increase the interaction accuracy [50] and allow even smaller targets.

However, manipulating digital objects by controlling a visual cursor on a touch screen does significantly reduce the immersion of directly interacting with the virtual world. Virtual pointers do not fulfill the direct touch manipulation paradigm.
5 Parallax error correction

![Image](image.png)

(a) 15 mm offset  (b) 7 mm offset

*Figure 5.10: Offset between the image plane and the interaction plane [180]*

5.4.4 Accurate interaction by hardware offset reduction

Reducing the offset between interaction plane and image plane is an effective approach to reduce the pointing error. In [180], the development of reducing the offset on a larger interactive screen as shown in Figure 5.10 is presented. It shows the offset between the LCD’s image plane and the interactive surface of the input device. In both cases, the tip of the pen contacts the interactive surface (a 6 mm glass pane), while the dot in the gray rectangle appears in the image plane underneath.

Studying the interaction quality with the Fitts’ law-based multi-directional tapping task [30] at two large digital whiteboards with different parallax offset, Kunz et al. [127] showed that the interaction quality increases with a reduced offset between interaction and image plane. The results clearly indicate the influence of the offset onto the interaction error, i.e. the parallax error. As such reduction can only be reached by modifying the underlying hardware, see Figure 5.10, it is naturally limited by the employed technology. A protective glass for example is necessary as it provides physical robustness of the interactive system, which is important for large interactive screens in public areas.

In order to further decrease the parallax error, additional software changes are required. Kent et al. proposed a non-linear correction, which takes into account the sensor noise of the tracking system [119] and Hong-Ping et al. presented a learning calibration method based on back-propagation neuronal networks [222]. A simpler approach to overcome the effect of the interaction errors is to avoid the need for accurate interaction by increasing the size of the targets.

5.4.5 Linguistic target reduction

On today’s touch sensitive phones and tablet devices, a widely used approach to reduce the need for precise interaction is reducing the number of targets. This principle increases the distance between targets and its interaction detection area. Assigning the target closest to the interaction position allows dealing with less accurate interaction precision.

Based on the linguistic context, i.e. the text structure or the word spelling, the computer reduces the number of valid targets. As an example, a text processor allows only selecting regions between words by touching the interactive screen. This is reasonable under the assumption that each word is typed in correctly, which is mostly assured by the automatic spell correction. If, however, the user wants to change a single insetted character, the cursor can only be put after the word and instead of next to the infix characters. The selection of single characters of a text document is not possible.
The keys of a soft-keyboard, which is a keyboard shown on the interactive screen to support text input, are closely arranged, small targets by definition. This does not allow reducing the number of targets. Hence, the computer corrects the input using a dictionary. It does not detect a single key from the user interaction, but rather a set of keys and selects the most probable in the context of the typed word.

The linguistic context allows reducing the number of valid targets, the user is irritated due to the resulting limitation. Although touch interaction imitates natural pointing interaction, reducing the targets does not seem to be reasonable to the user. However, the target reduction is not applicable for tasks that do not provide restricting context, i.e. painting.

5.4.6 Avoiding the need for precise interaction

Increasing the size of the digital targets is a common way to avoid the need of accurate touch interaction. As shown in Figure 5.11(a), common keypads next to the display are used to avoid touch-based HCI and big targets on the screen are used to decrease the pointing error. Several studies investigated the effect of the target size on the interaction quality of touch sensitive surfaces: Finger touch both benefits and suffers from the nature of direct input, the ISO standard 9241-9 recommends the target size to be at least equal to the width of the index finger [30]. A study by Colle et al. claims that larger target sizes lead to a better performance on touch based numeric keypad [68]. Jin et al. [115] investigated the target size and spacing on interactive surfaces for elderly persons and suggests a minimum button size of 11 mm square. Several researches found that the error rate of a single click pointing event decreases with the size of the digital targets [158, 115, 131]. Large targets, however, limit the number of widgets being simultaneously shown on the screen (see Figure 5.11(b)), and impact the usability of the application by increasing the number of menu layers and screen pages. The example shows a subset of selectable items. Separating navigation buttons allows to scroll through the list. Since the list is not ordered, the user must randomly scroll through the list to find the target item.

To improve the pointing detection without increasing the size of the target, the zone for detecting an interaction for a target could be extended, such that even an interaction that does not take place directly above the target is assigned to the (closest) target. This approach does also limit the number of widgets.
shown at a time. Although the interaction is improved, avoiding the need of accurate pointing interactions does not increase the pointing accuracy.

### 5.4.7 Irritation from spacial displaced haptic feedback

The parallax error is a spacial displacement between the virtual target shown on the screen and haptic feedback, which the user perceives when touching the interactive surface.

In the context of virtual reality, spacial displaced feedback describes the effect of a displacement between the visual and the haptic feedback of an object. This effect occurs on interactive screens, since the user’s touch feedback for a virtual object differs from its visual appearance on the screen, due to the offset between image plane and interaction plane.

The touch interaction quality was studied by Kohli et al. [122] at a compatible setup with an interactive screen that has an offset between interaction plane and image plane. In the context of redirected touching in virtual environments they investigated the quality of touching objects in a virtual environment with displacement between visual and tactile feedback. The setup gave the user a visual output on a head-mounted display. A real plane was installed in front of the user that gives the user haptic feedback when touching the targets of the Fitts’ law multi-directional tapping test. To investigate the influence of displacement, they moved the real plane (for the haptic feedback) and not its visual (in the virtual reality).

This setup is similar to touch screens with an offset between interaction and image plane, since the interaction position, which gives the haptic feedback, and the visual position of the target at the display differ spacially. Measuring the difference between a setup with and without offsets, they found, that the mean interaction quality decreases significantly in terms of the Fitts’ law interaction quality criteria movement time $MT$ with an increasing offset between visual of the target and haptical feedback position, although the error rate and throughput did not significantly differ.

### 5.4.8 Evaluating the interaction quality of interactive systems

Evaluating the quality of interaction is done by evaluating the quality of coordination. The hand-eye coordination is usually evaluated with the Peg Board Test [209]. It is designed to evaluate motor dexterity and coordination of assembly line workers. Since the original test is implemented on a hardware board, it does not cover working on an interactive screen.

The Fitts’ law-based (ISO 9241-9) performance studies [85] are common for evaluating human-computer interfaces [60] in terms of human motor behavior. The test measures the difficulty of interaction tasks and the human rate of processing information. The complexity of the task in terms of target distance and touch accuracy is analog to the amplitude and the noise of an electric signal. Hence, the task difficulty (ID) can be deduced from Shannon’s Theorem [194] as $ID = \log_2 \frac{2d}{w}$ with target distance $d$ and target width $w$ in one dimension. MacKenzie [139] improved the definition of difficulty by:

\[
ID = \log_2 \frac{d}{w + 1} \tag{5.7}
\]

The relation between movement time $MT$ and difficulty of the task is defined linearly as

\[
MT = a \cdot ID + b \tag{5.8}
\]
with empirically detected parameters slope \( a \) and intercept \( b \). With a given \( ID \) and a measured \( MT \), the parameters can be determined by linear regression. According to [30], the throughput \( TP \) (also called \textit{index of performance (IP)})) is well suited to show the difference in performance of two interaction devices or techniques. It is given by:

\[
TP = \frac{ID}{MT}
\]

Initially, Fitts presented a test with two targets and considered only the horizontal dimension. The participants had to interact with alternating objects. This one-dimensional test was extended to a multi-dimension tapping test by positioning multiple targets in a circle and indicating the opponent of the last target to be next [30].

The design of Fitts’ Law studies is as follows. The compared tasks with different levels of difficulty in terms of controlled variable tuples \((d, w)\) are defined. The level of difficulty follows immediately from Equation 5.7. Each condition is tested multiple times and the completion time and interaction error rate is captured. The error rate is either defined as interaction error, given by the offset between interaction position and target position), or the percentage of unsuccessful trails to hit a target.


Hardware pointing devices for digital screens were compared by MacKenzie et al. [141]. They also investigated software selection techniques for touch pads using Fitts’ Law in [140]. A variant of Fitts’ Law is proposed by [189] to predict pointing times on interactive screens. Due to its simplicity, several single button tests are also popular [115, 147].

As described above, Fitts’ law tests are originally used to evaluate the quality of human-computer interfaces. Nevertheless, the tests can be adapted to evaluate and model the human interaction behavior on interactive screens. Modeling the user is motivated by correcting the parallax error on interactive screens with model based controllers.

In the next section, variants of Fitts’ Law are used to evaluate the user in terms of showing the influence of the parallax error on the interaction error, the influence of the previous interaction position onto the interaction error of the next interaction, the viewpoint position of the user while interacting with the system, and the focal point of area targets.

5.4.9 Extending the interaction space

The gaming industry works with large TV screens and special gaming controllers that provide indirect input. They extend the interaction from simple button controllers to user movement or posture detection. The Nintendo Wii-Controller [129] enables to detect the user’s movement in front of digital screens with a dedicated controller. The detection of natural user hand motions capturing orientation and velocity at the controller enables intuitive game control. To detect the user’s body without any additional markers or hardware, the Microsoft\textsuperscript{TM} Kinect\textsuperscript{TM} (Kinect) [5] or Asus’ Wavi Xtion [28] utilize only visual feedback. They provide 3-dimensional location information of users. For gaming, they usually take the location information to control a virtual avatar. Vision based 2D and 3D body pose detection in terms of face, eye, hand and finger detection was introduced by van den Bergh [212].

Two context-aware applications in the field of ubiquitous computing are introduced by Antifakos [37]. The machine perception in terms of reacting on high-level intentions of the user is based on low-level
measurement data.

Since the introduction of the Kinect in late 2010, it has been rapidly adopted as a research tool and for the development of numerous applications [186, 90]. Garstka et al. present how to track the users’ viewpoint with a Kinect to simulate 3-dimensional projection on a 2-dimensional digital screen by stimulating the user’s spacial immersion. Gallo et al. present a controller-free interaction to explore medical images [87] and Ren et al. present a hand gesture recognition with the Kinect sensor [174]. There is a rapidly increasing number of systems that benefit from the Kinect’s simplicity, also in other application fields. In rehabilitation for instance, there are many systems that use the capabilities of the Kinect [63, 188].

Recently, interactive surfaces and user tracking are combined, meaning that the integration of direct and indirect interaction makes a system even more intuitive to handle. This extends the interaction space from direct manipulation on the touch panel to the real world space in front of or above digital surfaces (see Figure 5.3(a)).

Most of those systems address the idea of gesture recognition. Marquardt et al. [142] proposed a system that is able to fluently go from direct touch interaction to gesture recognition and therefore stated the term “continuous interaction space”. The system uses a glove with markers tracking the user’s hand with a vision based outside-in system. Although the tracking is highly accurate, wearing a glove makes the interaction not applicable for daily application. Also commercial systems like Microsoft Pixelsense [9] nowadays allow markerless tracking not only directly on the screen, but also slightly above it.

5.4.10 Research gap

The literature shows that high precision touch interaction is crucial on interactive screens. It, however, lacks of adaptive correction controllers to overcome errors that stem from dynamic behavior of the user, in particular the parallax error.

Section 5.3.1 shows that the parallax error depends on the user’s viewpoint. Hence, the parallax error can be corrected if the user’s viewpoint is directly captured.

In general, accurate 3-dimension head tracking does require an expensive tracking device with complex setups (i.e. vision based, inside-out, marker based tracking systems), which makes it realizable only in laboratory environments. To integrate the parallax correction feature also in affordable mass products, the upcoming and even more interesting question is:

- Is it possible to correct the parallax error without any additional hardware which tracks the user directly, and instead get use of the previous interaction (position and error) from the interactive system to estimate the current viewpoint?
- How can the user’s irritation be considered, which stems from the changing parallax correction due to a changing viewpoint estimate, in the estimation process?
- How to deal with changing users?

Model-based Bayesian filtering, like the well-known Kalman Filter [117] or Particle Filter [164], provide a well-studied and widely applied adaptive estimation of the real system state from noisy measurements. To develop a continuous correction controller that estimates the user’s viewpoint from interaction events, the user’s behavior must be modeled in terms of movement (process model) and correlation between interaction and viewpoint (observation model).

The viewpoint is given as 3-dimensional point in the space in front of the screen. The interaction error on the screen, however, is defined as two points: The interaction position on the interaction plane and the
target position on the display plane. Hence, the correlation between the viewpoint and the interaction is given as a straight line, which is used to estimate the user’s viewpoint from the interaction on the screen. However, the estimate (a line) is under-determined to define the viewpoint. Moreover, estimating the user’s viewpoint contains uncertainty due to the user’s dexterity.

Correcting the parallax error by estimating the user’s viewpoint may cause a rapidly changing interaction error correction. To additionally take the user irritation into account and to consider optimal control with limited correction gains, model predictive controller (MPC) consider also the future effects of the control actions onto the system (the user) to deduce a non-myopic control policy. Common MPC controllers apply planning out the future system behavior with respect to the process model. They consider the accumulated costs of applying an action instead of just the immediate costs. MDP controller allow modeling the process stochastically. MDP controller apply planning under uncertainty, since the effect of an action is modeled with respect to uncertain outcomes. For a given action, the process is modeled as Markov chain. In addition to the long term effect of control actions on the system development, POMDP controllers also consider the uncertainty due to sensor noise. The planner simulates the system control loop by alternately applying actions and observations to process a step into the future. Considering all the aspects leads to highly complex models and high efforts in computing the control policy (planning).

The upcoming question is: How to model the system dynamics, the sensors and the actions appropriately, such that the given problem is small enough to be apply POMDP controller (engineering aspect).

To deduce a POMDP model for correcting the parallax error from measurements, the user’s behavior in front of interactive screens is investigated in the next section. The models are used to design Bayesian filters estimating the user’s viewpoint from interaction errors and to develop non-myopic control strategies to correct the parallax error.

### 5.5 Human user behavior identification

In Section 5.3, it is proven that the parallax error exists. Since the parallax error depends on the user’s viewpoint, a correction controller can overcome the error for a given viewpoint and interaction position. However, the viewpoint position is not measured directly. Instead, it is estimated by a model based filter for which the underlying models are empirically deduced from studying the user’s behavior in front of interactive screens. ‘Inferring models from observations and studying their probabilities is really what science is about. [...] System identification deals with the problem of building mathematical models of dynamic systems based on observed data from the system.’ [136].

Conceptually, the user’s viewpoint is estimated by a Bayesian filter as introduced in Section 2.5.1 from interactions on the screen’s surface. As shown in Equation 2.14, an estimate (or belief) of the system state $s$, which represents the user’s viewpoint, is updated by an observation $o$ which is given by the interaction error on the screen and a process update which describes the system dynamics of the state.

The observation update is deduced from the observation model $P(o|s)$ utilizing Bayes’ theorem and the (independent) prior probability for observations $P(o)$ and states $P(s)$. However, $P(o)$ can be directly deduced from the prior state and the correlation between observation and state using marginalization: $P(o) = \sum_{s \in S} P(o|s)P(s)$ (see Equation 2.14). Hence, the observation model is given by $P(o|s)$ and $P(s)$. The second aspect of the Bayesian filter is the process update. The stationary process model describes the development of the system state over time which is - in case of the user’s viewpoint - the user’s movement (or viewpoint dynamics) within a certain time step. The probability of being in state $s'$ at the next time step for a given state $s$ and correction action $a$ expresses the user’s movement as $P(s'|s, a)$, since the parallax error is corrected for the next interaction.
5 Parallax error correction

The state space and the corresponding observation space are defined in Section 5.5.1. In section 5.5.1, the prior probability of the user’s viewpoint $P(s)$ is deduced from analyzing the independent viewpoint position in front of the screen. On large screens, the position depends on the targets on the screen due to the limited arm length of the user. To apply a common target distribution investigating the user behavior, the interaction position on the screen is analyzed in Section A.2.

The transition model describing the user’s position change in front of the screen is given by $P(s_{t+1}|s_t)$ is deduced from measurement of tracking the subject’s viewpoint with high quality tracking devices in a lab environment, while using an interactive system. The control loop time step $i$ is determined from the average interaction frequency in Section A.2.

The observation model in terms of the probability distributions for $P(o|s)$ is developed considering two aspects. The haptic error, that stems from the user’s dexterity, is inferred from correlation between viewpoint and interaction (error) for small targets in Section 5.5.3. The interaction error is given as offset between target point and interaction point in screen coordinates. However, common widgets are almost exclusively area targets (i.e. buttons). To provide a reference point for calculating the interaction error, the focal point of area targets is investigated in Section 5.5.5. The resulting observation model combines the uncertainty caused by the user’s dexterity and the area target focal point.

All models are empirically deduced from measuring users with high quality tracking devices in a lab environment, while using an interactive system. Since identifying the human behavior is affected by uncertainty, the resulting models are stochastic. Counting the occurrence of a sampled random event is transformed to a probabilistic model. To this end, the human behavior is mostly identified in an open loop way. The interactive system does not provide any feedback of the interaction quality to the user to not affect his interaction behavior, i.e. the Fitts’ Law tapping tests show the next target even if the user did not hit the last one. Although the user might find out that he does not need to hit the given target, not giving feedback is more important to not allow the user to compensate the parallax error self-initiated.

5.5.1 User viewpoint location

In this section, user’s behavior is analyzed in terms of the position in front of the screen. (A confirmation study is documented in Section A.3.) The question of the user’s typical position should be answered. To set up a model that matches the user’s behavior closely, a user study is conducted, which tracks the position of the user’s head (viewpoint) in front of a large digital whiteboard as described in Appendix A.1.

The following analysis is based on measurements of the user study described in detail in Appendix A.1. The accuracy of capturing the visual marker is evaluated to be $\sigma = 0.867$ [mm]. The dimensions are referred as follows: The x-dimension is the horizontal dimension parallel to the surface of the screen, increasing to the right; the y-dimension is the vertical axis parallel to the screen increasing to the floor; and the z-dimension is defined to be orthogonal to the screen. The coordinate system’s origin is defined to be at the upper left corner of the display.

The coordinate system can be expressed with respect to two origins: Either to the screen, which leads to a static coordinate system, or the relative viewpoint $V$ w.r.t. the (changing) target (point) $T$, as shown in Figure 5.12. A good transition model, however, contains a low amount of uncertainty. If the relation between target location and the user’s viewpoint is more stable than the viewpoint position w.r.t. the screen, the relative viewpoint is preferable. Additionally, if the viewpoint is defined relative to the target’s origin, the model gets smaller (with the same resolution) due to the limiting arm length of the user, which is smaller than the size of large interactive screens.
5.5 Human user behavior identification

The relative user viewpoint is assumed to be more stable than the position with respect to the screen’s origin, since the user moves according to the target. Moreover, it is assumed, that the relative viewpoint is symmetrically distributed around the perpendicular screen position of the viewpoint [149]. It is moreover assumed that the viewpoint of a left-handed (right-handed) subject is located right (left) of the target location. In any case, the three degrees of freedom of the user’s viewpoint are assumed to be independent and, moreover, Gaussian distributed.

**Measurement analysis**

In this section, the viewpoint position stability is analyzed in terms of the variance w.r.t the screen and target origin.

As summarized in Table 5.2(a), the distance between screen and user is measured between 476 mm, which corresponds the median forearm length of 417 mm [153], and 759 mm, which corresponds to the average arm length of 722 (690) mm of men (women) [73]. With a mean value of 577 mm the distance to the screen corresponds 80% (84%) of the average arm length. Hence, the subjects do not touch with arm up. With a variance of 42 mm, the distance is as stable as the vertical position.

**Screen origin** The histograms from measuring the user’s viewpoint are shown in Figure 5.13 and the probability distribution characteristics listed in Table 5.2(a). They show that on average the user is located symmetrically around a mean value of 431 mm, which is located slightly right of the center of the screen at app. 310 mm. The high variance in horizontal direction shows that the viewpoint heavily differs and indicates a strong movement in horizontal direction, which is analyzed in detail in Section 5.5.2. In vertical direction, the viewpoint is more stable ($\sigma = 41$ mm) around the offset of 229 mm, which indicates less movement in vertical direction.

The Pearson correlation coefficient between the horizontal and vertical dimension is -0.22. Between the horizontal (vertical) and the distance to the screen it is 0.011 (-0.18). Hence, independence between the three degrees of freedom of the viewpoint location can be assumed.

The data is tested to be Normal distributed two-side $\chi^2$-test. The p-values of the user’s position in horizontal and vertical direction as well as the distance to the screen are $1.91 \cdot 10^{-112}$, $3.50 \cdot 10^{-7}$ and 0.
5 Parallax error correction

Figure 5.13: User viewpoint position relative to the screen [mm] (histogram)

Figure 5.14: User viewpoint distance to screen [mm] (histogram)

Figure 5.15: Offset between viewpoint and interaction point [mm] (histogram)
### 5.5 Human user behavior identification

#### (a) Screen Origin

<table>
<thead>
<tr>
<th></th>
<th>Minimum</th>
<th>1st quantile</th>
<th>Mean (median)</th>
<th>3rd quantile</th>
<th>Maximum</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal</td>
<td>59</td>
<td>352</td>
<td>431 (419)</td>
<td>503</td>
<td>780</td>
<td>116</td>
</tr>
<tr>
<td>Vertical</td>
<td>108</td>
<td>201</td>
<td>229 (229)</td>
<td>259</td>
<td>351</td>
<td>41</td>
</tr>
<tr>
<td>Screen distance</td>
<td>476</td>
<td>550</td>
<td>577 (572)</td>
<td>597</td>
<td>759</td>
<td>42</td>
</tr>
</tbody>
</table>

#### (b) Target Origin

<table>
<thead>
<tr>
<th></th>
<th>Minimum</th>
<th>1st quantile</th>
<th>Mean (median)</th>
<th>3rd quantile</th>
<th>Maximum</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal</td>
<td>-426</td>
<td>-255</td>
<td>-122 (-152)</td>
<td>-23.97</td>
<td>354</td>
<td>179</td>
</tr>
<tr>
<td>Vertical</td>
<td>-412</td>
<td>-229</td>
<td>-69 (-49)</td>
<td>108</td>
<td>219</td>
<td>178</td>
</tr>
</tbody>
</table>

**Table 5.3: Viewpoint characteristics [mm]**

**Target origin** The relative viewpoint is measured as offset between interaction position on the screen and the user’s viewpoint. A negative value indicates the viewpoint to be located left of the target, a positive value indicates the user’s viewpoint right of the target.

The measurement results are listed in Table 5.2(b) and shown in Figure 5.15. The high variance in horizontal and vertical direction indicates a heavily changing value.

The relative viewpoint shows a mean value of -122 mm in horizontal direction. Since the 3rd quantile is also negative, the user is mostly located left of the target location. The vertical position shows a mean (median) value of -69 (-49) mm. This indicates the user’s viewpoint above the target position, which is reasonable for the setup of the study.

The horizontal and vertical measures show a similar standard deviation of 179 and 178 mm. The range of the offset between viewpoint and target location (horizontally 781, vertically 631 mm) is higher compared to the screen origin measures (horizontally 721, vertically 243 mm). This indicates, that the user’s viewpoint does not change according to the interaction position, which occurs for example, when the target moves but the user does not change his position.

The gradient of the histogram in Figure 5.15(b) of the horizontal measures indicate less variance for smaller percentiles: 50 % of the horizontal values are located within a range of 310 mm between -255 and -24 mm, while the corresponding vertical interval is given by [-229,108] mm. The almost equally distributed histogram for the vertical dimension (see Figure 5.15(a)) indicates that the user does not change the viewpoint position according to the target position. In fact, the distribution seems to correlate with the independent probability distribution of the sampled target position of the user study task.

Testing the data being Normal distributed with the two-side $\chi^2$-test resulted in the following p-values: 1.25e-76 for the data in horizontal dimension and 0.0 for the data in vertical dimension.

**Effect of left and right-handed user onto the viewpoint** Table 5.4 shows that the viewpoint position relative to the target differs for left- and right-handed subjects. Although the high standard deviation around 170 mm indicates a strong movement independently from the preferred interaction hand, the measures confirm the initially expressed statement: The viewpoint of left handed people (who interact mostly with the left hand) is commonly located right of the interaction position. The measures
5 Parallax error correction

<table>
<thead>
<tr>
<th></th>
<th>Minimum</th>
<th>1st quantile</th>
<th>Mean (median)</th>
<th>3rd quantile</th>
<th>Maximum</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right-handed</td>
<td>-235</td>
<td>15</td>
<td>121 (89)</td>
<td>206</td>
<td>554</td>
<td>163</td>
</tr>
<tr>
<td>Left-handed</td>
<td>-426</td>
<td>-255</td>
<td>-122 (-152.35)</td>
<td>-23</td>
<td>354</td>
<td>179</td>
</tr>
</tbody>
</table>

*Table 5.4: Horizontal viewpoint characteristics relative to the target [mm]*

show that the centered 50 percent quantiles do not overlap. Hence, left-handed people are distinguishably located right of the target on the screen and right-handed people located left respectively.

Conclusion

The state space is defined as the user’s viewpoint and the unconditioned probability of the state is used as prior distribution estimating the user’s viewpoint. A probabilistic model, however, is expressive if the variance and the range (the uncertainty) is low.

As stated in Hypothesis 2, it is intuitively clear that the position of the viewpoint must change at a large interactive screen, due to the limited arm length of men (women) that ranges from 662 (616) mm for the 5. percentile to 787 (762) mm for the 95. percentile [73, 74]. The measurements show that the user does change the viewpoint position in front of the screen. However, the measurement does not indicate a strong correlation between the viewpoint and the target in terms of a low variance of the relative viewpoint. The higher variance and range of the viewpoint relative to the target compared to the viewpoint relative to the screen indicates that the user’s viewpoint behaves more stable w.r.t. to the screen’s origin than to the target. The user mostly stays within $\pm42$ [mm] around the average position. Hence, the prior distribution $p(s)$ would be more expressive w.r.t. the screen’s origin. It is, however, reasonable to assume that this effect might be even more notably on small displays, since the user is not forced to move in order to reach targets physically.

In terms of the viewpoint distance to the screen, the user’s arm is bended to app. 80 % of the full arm length to interact on the screen. Due to the high variance of the viewpoint location relative to the target, the user seems to neither make a step towards the target in horizontal dimension nor bend down to hit a target in the lower section of the screen. The approximately equally distributed measure of the relative viewpoint in vertical dimension does not carry any prior information for the Bayesian filter.

Also due to ergonomic reasons, right (left) handed people mostly stand left (right) of the interaction position, and the target position close to it. Hypothesis 3 is thus confirmed. The subjects are distinguishably located left (right) of the target w.r.t. their primary hand.

All measurements were tested to be Gaussian distributed. The resulting p-values of less than 0.01 show that the data is not Gaussian distributed for a significance level of 0.1% independent from the origin.

Measuring the user’s viewpoint allows modeling the prior distribution $P(s)$ of the Bayesian filter to estimate the user’s viewpoint from screen interactions (observations). In the next section, the process model will be deduced from empirically studying the user’s movement in front of the screen.

5.5.2 User viewpoint dynamics

In section 5.5.1, it is found that the user’s position in front of the screen changes over time while using a large interactive screen. In this section, the movement is investigated more in detail. (A confirmation
5.5 Human user behavior identification

study is documented in Section A.3.) Since the Bayesian filter updates the estimate with a process model that expresses the system dynamics, the viewpoint (system state) is investigated in terms of discrete time movements.

The question where he will probably move in the next time-step $t_{i+1}$ step relative to the current position at $t_i$ should be answered. Since the correction controller will use screen interactions as observations for the system state, the according time step of the discrete time process model can be deduced from the average time between two interactions. This is $1.32 \text{ s} \ (\sigma = 0.40 \text{ s})$ for the study described in Appendix A.1. Hence, the user’s viewpoint movement is expressed respectively.

To set up a model that matches the user’s behavior closely, a user study is conducted, which tracks the position of the user’s viewpoint in front of a large digital whiteboard as introduced in Appendix A.1.

It is assumed that the user will mainly move in the horizontal direction, but very little in the vertical direction and the distance to the screen. It is, moreover, assumed that the next viewpoint location of the user is symmetrically and Gaussian distributed around the current location.

### Measurement analysis

The measurement results are summarized in Table 5.5. Figure 5.16 shows the user movement. The histograms show the user’s viewpoint position changes in all three degrees of freedom (horizontally, vertically and the distance to the screen) w.r.t. the display orientation. A negative value in the horizontal (vertical, distance-to-screen) direction represents movement to the left (upwards, away from the screen) and a positive value indicates movement to the right (downwards, towards the screen).

The horizontal measures show a slightly positive mean value of 7 mm and the average value of the vertical measures have a mean value of 8.16 mm. The median value is in both cases equal (and close) to zero. The fastest horizontal movement shows a distance of 388.71 mm within 1.32 s, which corresponds to $0.3 \text{ m/s}$. The fastest absolute movement in vertical (distance-to-screen) direction is 128.87 (176.03) mm and thus significantly lower.

The horizontal movement shows a high variance ($\sigma = 99.57$). In the vertical dimension and the distance to the screen, the variance is significantly lower (see Table 5.5). Figure 5.16 shows that the shape of the histogram is symmetrically distributed around the median value.

The Pearson correlation coefficient between the horizontal and the vertical movement is -0.29 and between the horizontal (vertical) and the distance to the screen is 0.11 (-0.11).

The p-value of testing the movement data against the Normal distribution with the $\chi^2$-test are $9.35 \cdot 10^{-6}$ for the horizontal and $2 \cdot 10^{-3}$ for the vertical movement, and $2.44 \cdot 10^{-13}$ for the change of the distance to the screen.

### Table 5.5: Viewpoint movement characteristics [mm]

<table>
<thead>
<tr>
<th></th>
<th>Minimum</th>
<th>1st quantile</th>
<th>Mean (median)</th>
<th>3rd quantile</th>
<th>Maximum</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal</td>
<td>-303.65</td>
<td>49.94</td>
<td>7.09 (-1.26)</td>
<td>59.97</td>
<td>388.71</td>
<td>99.57</td>
</tr>
<tr>
<td>Vertical</td>
<td>-100.69</td>
<td>-21.18</td>
<td>0.08 (0.51)</td>
<td>15.73</td>
<td>128.87</td>
<td>34.43</td>
</tr>
<tr>
<td>Screen Distance</td>
<td>-176.03</td>
<td>-15.35</td>
<td>-0.48 (-1.01)</td>
<td>14.63</td>
<td>148.47</td>
<td>32.81</td>
</tr>
</tbody>
</table>

83
Conclusion

Examining the user’s viewpoint in front of a large interactive screen *DigiBench* showed that the user moves mainly in horizontal direction, although the peak around zero movement clearly indicates the inertia of the motion. As already assumed from the time-independent position measurements (see Section 5.5.1), the subjects move very little in the vertical dimension and in distance to the screen within one time step (see Figure 5.16(b)).

The measures indicate the movement being distributed symmetrically around the previous location in every direction. Based on the measurement results, a process model probabilistically describes the next viewpoint $s'$ position within the given discrete time step as $P(s'|s)$.

The assumption that the user’s movement is Normal distributed is dropped, since the p-values of the $\chi^2$-tests are below 0.01 (significance level of 0.1%).

5.5.3 Correlation between interaction and viewpoint

As motivated in Section 5.4.10, the parallax error can be corrected by estimating the user’s viewpoint in front of the screen from the interaction error on the screen. It is assumed that the interaction position corresponds to the target position on the screen with respect to the parallax error. As introduced in Section 2.5.1, a Bayesian filter models the prior distribution of the system state $P(s)$, its dynamics $P(s'|s)$ and the correlation between observations (in this case the interaction error) and the system state (the user’s viewpoint) as conditional probability distribution $P(o|s)$.

In this section, the first aspect of the observation model is deduced from the correlation between the user’s actual viewpoint and interaction error on the digital surface for small targets. The interaction error is given as offset between the center of the target and the actual location of the interaction. It is assumed that the measures contain uncertainty which is caused by the user’s dexterity, as discussed in Section 5.3.2. Additionally, smoothing the observation data might increase the correlation between the state and the observation. The measurements are deduced from a user study which investigates the interaction behavior on a large interactive screen. The study is described in detail in Appendix A.1.

The correlation analysis considers the interaction error and the relative viewpoint, which is given as offset between the target (point) $T$ and the viewpoint location $V$ (see Figure 5.17). The offsets are normalized with respect to the z-distance and treat the horizontal and vertical dimension separately.
5.5 Human user behavior identification

Figure 5.17: Dexterity model

Measurement analysis

Figure 5.18 shows the histogram of measurements on the normalized and discretized relative viewpoint and interaction error space to the interval $[-1, 1]$ with a discretization of $\frac{1}{10}$, which is normalized by the z-distance. Each bucket represents the number of measurements within the given viewpoint and interaction error interval by gray scale. With regards to the appropriate normalization, a row indicates the probability distribution over the viewpoint space $S$ for a given observation $\bar{o}$: $P(s | o = \bar{o})$ and, vice versa, a column the probability distribution over the observation space $O$ for a given viewpoint $\bar{s}$: $P(o | s = \bar{s})$. The resulting model is expressed discretely due to the discrete nature of the interaction coordinates of the interactive system.

The horizontal measures show a high density around -0.3 (see Figure 5.18(b)), which corresponds to the results in Figure 5.15(b).

As already mentioned, the measurements in vertical dimension do not indicate a high correlation between $S$ and $O$. The measurement results of the relation between interaction and viewpoint from Section 5.5.3 indicate a strong correlation in the horizontal dimension. The width of this distribution over the observation space indicates the certainty of evidence for the observations.

Figure 5.19 shows scatter plots (with 5505 measurement points) of the relation between viewpoint and interaction error in vertical and horizontal dimension.

Since the Pearson’s correlation coefficient given a value of 0.806 (0.01) for the horizontal (vertical) data, the horizontal data is globally positively correlated and the vertical data is not correlated. Hence, only the horizontal data was analyzed in more detail. The linear model in Figure 5.19(b) indicates that the correlation is globally biased by a normalized factor of -0.3. The Loess model [67, 66] for local regression shows an almost linear trend over the entire space: The data sets are positively related, meaning that the interaction occurs right (left) to the target if the user stands right (left) of the target. The area between -0.5 and 0.0 (the viewpoint) indicates a correlation of higher order with a medium correlation coefficient of 0.462. Linear regression of the horizontal data is parametrized with a gradient of 0.90 and a standard deviation of 0.012.

Figure 5.20 shows the change of correlation for smoothed data by several smoothing window functions. Although the correlation factor increases with a window size of 3 and 4, the overall value stays low. Independently from the applied window type, smoothing the horizontal data significantly decreases the
5 Parallax error correction

Figure 5.18: Frequency of the relative viewpoint and interaction error (thermal plot)

Figure 5.19: Normalized relative viewpoint and interaction error (scatter plot)
5.5 Human user behavior identification

![Graphs showing correlation between window size and Pearson correlation coefficient for flat, Hanning, Hamming, Bartlett, and Blackman windows.](image)

(a) Vertical  
(b) Horizontal

*Figure 5.20: Smoothed correlation between relative viewpoint and interaction error*

Correlation between observations and estimated system state.

**Conclusion**

The interaction error shows a good correlation to the user’s viewpoint in horizontal dimension. However, the vertical measurements do not correlate. Smoothing the interaction error does not increase the correlation.

The measurements motivate to deduce an observation model, which describes the probability of getting an interaction error (observation) given a viewpoint (state) in horizontal dimension. From this measurements, estimating the vertical position of the user is not particularly promising.

So far, the correlation between viewpoint and interaction error is investigated for small targets, since the interaction error can only be calculated from a target and interaction point on the screen.

**5.5.4 Interaction precision**

Lee and Zhai stated that targets for touch-sensitive screens should be at least the size of 22 mm in diameter, which corresponds to the user’s fingertip [131]. Intuitively, this seems reasonable since the user can not see small targets behind his finger touching them at an interactive screen. However, a more accurate interaction should be possible with a pen (tangible user interface) providing a fine tip.

**Hypothesis 4.** On interactive surfaces, pen interaction is significantly more accurate than bare finger interaction on hitting small area targets.

In a user study, which is described in Appendix A.4, the interaction precision for the subjects pointing at targets with a) their bare finger and b) a pen TUI is analyzed. The interaction precision is measured as interaction error which is defined as offset between the interaction location and the target center. In the following, the measurement results are discussed.
5 Parallax error correction

Figure 5.21: Interaction error with bare finger and pen TUI [mm] (histogram)

<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>1st quantile</th>
<th>Mean (median)</th>
<th>3rd quantile</th>
<th>Max</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Horizontal</td>
<td>Pen</td>
<td>18.23</td>
<td>-4.29</td>
<td>-0.63 (-1.07)</td>
<td>3.21</td>
<td>15.01</td>
</tr>
<tr>
<td></td>
<td>Finger</td>
<td>-22.52</td>
<td>-5.36</td>
<td>-1.32 (-1.07)</td>
<td>2.14</td>
<td>16.08</td>
</tr>
<tr>
<td>Vertical</td>
<td>Pen</td>
<td>-34.74</td>
<td>-12.12</td>
<td>-9.44 (-9.69)</td>
<td>-6.46</td>
<td>10.50</td>
</tr>
<tr>
<td></td>
<td>Finger</td>
<td>-31.51</td>
<td>-12.12</td>
<td>-8.69 (-8.88)</td>
<td>-5.65</td>
<td>9.69</td>
</tr>
</tbody>
</table>

Table 5.6: Interaction error with bare finger and pen TUI [mm]

Measurement analysis

The horizontal and vertical interaction error is shown in Figure 5.21 and summarized in Table 5.6. The results show no significant difference between the two pointing methods bare finger and pen TUI neither in vertical nor in horizontal direction. The distributions show similar mean and median values. The variances are similar and the center 50% quantiles are not separable.

The horizontal (vertical) hit rate with the pen TUI is 75% (24%) and the horizontal (vertical) hit rate with just the finger is 79% (28%).

Conclusion

The hit rate for the given target is, in contrast to the assumption, even higher than by using the pen. Comparing the interaction error for a target of $12 \times 13$ mm on a large interactive screen does not show significant differences between pointing with the user’s bare finger or a pen TUI. Although the pen TUI provides a fine tip, the pointing accuracy does not increase.

It is concluded that the pointing quality is similar for interacting with the bare finger and a pen TUI at the given setup. Hypothesis 4 is thus dropped.

In the next section, the influence of the area onto the focal point of targets is investigated.
5.5 Human user behavior identification

![Area widget on screen](image)

**Figure 5.22:** Detection zone of small area targets (widgets)

(a) Push button  
(b) Checkbox

**Figure 5.23:** Geometrical center of GUI widgets

### 5.5.5 Focal point of area targets

As stated in Section 5.4.10, the parallax error is corrected estimating the user’s viewpoint in front of the screen from the interaction error on the screen. In Section 5.5.3, it is shown that the interaction error correlates with the user’s viewpoint for a small target. However, GUI targets are usually not small. To estimate the user’s viewpoint from the interactions on common GUI systems, the focal point must be defined as a reference point for the interaction error.

For static calibration, a common way to motivate the user in the learning phase to hit a certain point is to show a tiny target or using a cross hair pointing at the center of the target widget, as shown in Figure 5.8(a). Hence, the interaction error is clearly defined as distance between the interaction position (which is given by the touch panel) and the target center. However, this technique is only applicable if the system is statically calibrated during a dedicated calibration process, which is common for interactive systems. If the correction controller deduces the correction parameters while the user interacts with the system, it must analyze common application widgets to deduce the pointing error, as shown in Figure 5.22. Unfortunately, common GUI applications widgets, like buttons, do not support a single target point. Hence, the focal point is deduced for area widgets.

**Definition 12.** The **Geometric center** of an area widget is defined as the median point of the bounding polygon of the widget, as shown in Figure 5.23.

**Definition 13.** The **Optical center** of an area widget is defined as the center point of the content. As shown in Figure 5.24, the Optical center of a text button widget is expected at the center of the text, which is not necessarily coherent with the Geometric center due to the text alignment.

**Definition 14.** The **Focal point** of a target is the point within the area of the GUI widget which the user aims to hit.
The correction controller can only correctly estimate the viewpoint from the interaction error if the aimed interaction location is assumed correctly. Hence, the focal point of area targets is investigated in detail under the following assumptions:

**Hypothesis 5.** The user interaction is next to the Optical center of the widget and not on the Geometric center. The interaction error is Normal distributed.

**Hypothesis 6.** The deviation of the interaction error is constant, independent of the target’s size.

**Hypothesis 7.** The interaction position depends on the user’s aimed hit location.

The experiment design and setup is described in detail in Appendix A.5 and in [173].

**Measurement results**

The following section summarizes the results of measuring the interaction error for different widget types. First, the focal point is investigated. The average interaction offset from the center of the target indicates if the user aims for the Geometrical or the Optical center of the widget, respectively. Then, the influence of the widget size onto the interaction error is investigated. Finally, the aimed interaction points are compared to the actually measured interaction locations.

To analyze the influence of the Optical center in terms of the content location onto the interaction error, the measurements for three different symmetrical push buttons are compared (see Figure A.13(b)). The button text is either located at the upper left, the center or the lower right corner of the target area. The resulting interaction is normalized by the widget size as shown in Figure 5.25. The interaction error of wide push buttons (see Figure 5.25(a)) is not Normal distributed (left: $p = 4.51 \cdot 10^{-5}$, center: $p = 2.2 \cdot 10^{-16}$, right: $p = 0.01$). Although the influence of the Optical center is shown in the box plot, the differences are not significant. The same holds true for symmetrical widgets in horizontal (left: $p = 0.02$, center: $p = 0.0004$, right: $p = 7.19 \cdot 10^{-5}$) and vertical dimension (top: $p = 0.02$, center: $p = 0.006$, bottom: $p \ll 0.01$).

The box plots, shown in Figure 5.25, show that the interaction position does not significantly differ for different optical centers, although the users tend to interact at the optical center. The same holds true for larger buttons. Due to the overlapping of the 50% quantiles. It is concluded that the optical center does not affect the interaction position on rectangular push buttons. Hence, the Optical center does not influence the interaction.

The distributions of the interaction error on push buttons, links and checkboxes are summarized in Figure 5.26. The results show that the interaction error on push buttons and links is symmetrically distributed around the geometrical center. The Anderson-Darling test does not confirm that the data is normally distributed ($p \ll 0.01$). In contrast, the measures for the checkboxes are not symmetrically distributed around the Optical center (on the left).

The correlation between the widget size and the deviation of the interaction error is shown for push buttons in Figure 5.27 and for links in Figure 5.28. The analysis shows that the deviation of the interaction...
Figure 5.25: Influence of optical center onto interaction error for buttons

Figure 5.26: Interaction error on different widget classes

error increases linearly with the size of the widget. Moreover, the expected value is stable, independent from the widget size. Figure 5.27(a) shows the horizontal interaction error for rectangular push targets with a height of 18 px and dynamic width; In Figure 5.27(b) the offset is normalized by the target width. The static behavior of the 50 % center quantile shows a stable coherence between horizontal widget size and interaction offset. The same holds true for link widgets, as shown in Figure 5.28. The Anderson-Darling test of analyzing the links showed that the interaction error is normally distributed for links wider than 16 px (see Table 5.7 (p > 0.05).

As part of the questionnaire, the subjects were asked to sketch the aimed interaction position for different widgets (see Figure A.14). The user did either aim for the Optical center or for the Geometrical center exclusively. Moreover, the user stick to their decision for all widget types. Table 5.8 lists the aim for the Geometrical center in percent. All subjects state to aim for the Geometrical center of link and quadratic push buttons, which corresponds to the measurement results. The same holds true for 95% of the subject for rectangular buttons. Only 77 % of the subjects aim for the Optical center of the checkbox widget which also confirms the interaction measurements.

Conclusion

In this section, the interaction behavior for different classes of widgets was investigated.

The measurements indicate that the focal point for push buttons and links is located at the Geometrical center of the target. The alignment of the button’s label text does not influence the hit point. For checkboxes the focal point is located at the Optical center of the widget. Hypothesis 5 is dropped for link and
5 Parallax error correction

![Diagram](image1)

**Figure 5.27:** Correlation between stability of horizontal interaction error and button size for buttons

![Diagram](image2)

**Figure 5.28:** Correlation between stability of horizontal interaction error and target size for links (height 18px)

<table>
<thead>
<tr>
<th>Link width [px]</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0.01</td>
</tr>
<tr>
<td>39</td>
<td>0.10</td>
</tr>
<tr>
<td>51</td>
<td>0.47</td>
</tr>
<tr>
<td>66</td>
<td>0.77</td>
</tr>
<tr>
<td>80</td>
<td>0.30</td>
</tr>
<tr>
<td>110</td>
<td>0.18</td>
</tr>
<tr>
<td>200</td>
<td>0.2</td>
</tr>
</tbody>
</table>

**Table 5.7:** Anderson-Darling test for normal distribution

<table>
<thead>
<tr>
<th>Widget class</th>
<th>Aim for Geometrical center [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkbox</td>
<td>23</td>
</tr>
<tr>
<td>Link</td>
<td>100</td>
</tr>
<tr>
<td>Rectangular button</td>
<td>95</td>
</tr>
<tr>
<td>Quadratic button</td>
<td>100</td>
</tr>
</tbody>
</table>

**Table 5.8:** Aimed interaction position
button widgets. It is confirmed for checkboxes. Additionally, it is concluded that the interaction error for push buttons is not Normal distributed.

From the measurements of the interaction error for different widget sizes, it is concluded that the deviation of the interaction offset from the geometrical center coheres with the widget size. The deviation of the interaction distributions grows linearly with the size of the widget. Hence, the uncertainty of actually hitting the focal point (variance of measures) definition grows with the size of the target. Since the expected value is not affected by the widget size, it is concluded that the focal point (mean value of measures) of the widget is not affected by the widget size. Hence, Hypothesis 6 is dropped.

Moreover, it is concluded that the aimed interaction location corresponds to the measures of the interaction location. Hypothesis 7 is not dropped.

5.5.6 Conclusion

In the preceding section, the human behavior for working on interactive systems is characterized. Section 5.5.1 describes investigating the user’s viewpoint location in front of a large interactive screen. It is found that the user’s position mainly differs in horizontal direction and stays stable in vertical direction and in terms of the distance to the screen. It is, moreover, shown that the user moves less in relation to the screen than in relation to the GUI target location. It is reasonable to assume that the user will move even less on small screens, since potential targets on the screen are located within the reachable region limited by the user’s arm length. It is also found that left-handed (right-handed) subjects are located right (left) of the target in horizontal dimension. Analyzing the correlation between the 3 degrees of freedom of the user’s viewpoint location and movement shows that the dimensions can be assumed as independent. The distribution of the user’s viewpoint location will be used as estimate prior as described in the introduction.

Section 5.5.2 shows the results of investigating the user’s viewpoint in terms of movement. It is stated that the user moves mainly in horizontal direction with a speed up to $0.3 \text{ m/s}$. The measurement in vertical direction and in distance to screen shows less movement. In general, the movement analysis confirms the results in 5.5.1. Within 1.3 seconds, the next viewpoint location in time is symmetrically but not Normal distributed around the last location, which allows to set up regular transition model of the user’s viewpoint.

Since the parallax error correction controller estimates the user’s viewpoint from interaction error on the interactive screen, the correlation between viewpoint and interaction error is investigated in Section 5.5.3. The user studies show a good correlation in horizontal direction. Vertically, the interaction error does not correlate with the viewpoint location. Smoothing the interaction error, however, does not increase the correlation. The correlation of the user’s viewpoint and the interaction error was investigated for small targets. However, graphical user interfaces of common application do not necessarily provide small targets. To calculate the interaction error from area targets, the focal point is investigated as aimed target point in Section 5.5.5. The measurements show that the focal point is mostly located at the Geometrical center of the target and the deviation is distributed symmetrically around the Geometrical center. The deviation grows linearly with the widget size. To deduce the interaction error, the formulated focal point definition can be used as the target point reference for the investigated widgets. To utilize reliable target specifications, the correction controller should consider small targets. In Section 5.5.4, it is found that the interaction accuracy does not increase by using a pen TUI with a smaller tip compared to the user’s bare finger.

Hypothesis 2 is confirmed by the measurement results. In Section 5.3, it was stated that the parallax error
can not be overcome by static calibration, since it depends on the user’s changing viewpoint. The parallax error is shown to depend on the viewpoint in Section 5.3.1. In Section 5.5.2, it is shown by experiment that the user’s viewpoint does change over time. Thus, the parallax error can not be overcome by static calibration, which confirms Hypothesis 2.

The results in Section 5.5.1 confirm Hypothesis 3. The handedness can be deduced from measuring the interaction error. The measures show a significant difference in the viewpoint position for left- and right-handed people, respectively. Since the horizontal interaction error correlates with the viewpoint location, it can be concluded that left- and right-handed people are distinguishable by the sign of the horizontal interaction error.

In the following section, the measurement results of identifying the user’s behavior in front of interactive screens are transferred into model-based controller, correcting the parallax error on interactive screens.

### 5.6 Tracking correction controller

It is claimed that a dynamic correction of the parallax error is possible, when the user is tracked directly. The parallax error can be directly deduced from the viewpoint and the interaction point. As shown in Figure 5.29(a), the correct mapping of the interaction point \((I_{x,y})\) to the target point \((T_{x,y})\) is given by \(-I_x\) (see Equation 5.10).

\[
I_{x,y} = T_{x,y} \\
\Leftrightarrow 0 = I_{x,y} - T_{x,y} \\
\Leftrightarrow 0 = (T_{x,y} + I_{x,y}) - T_{x,y} \\
\Leftrightarrow 0 = I_{x,y} \\
0 = I_x + c \Leftrightarrow c = -I_x \\
-I_x = -\frac{V_x-o}{V_z+o} \tag{5.10}
\]

If, however, the current position of the user’s viewpoint \((V)\) and interaction at the screen \((I)\) are known, the correction controller determines the point, the user meant to interact with on the straight line between \(V\) and \(I\), with respect the offset of interaction plane and image plane. This approach eliminates the parallax error potentially perfectly, if the user interacts exactly in line of sight.

As shown in Figure 5.29(b), the tracking based controller corrects the parallax error by directly tracking the 3 dimensional position of the user’s viewpoint and 2 dimensional interaction point (preprocessor component). With respect to the offset of interaction plane and image plane, the controller maps the interaction to the display coordinates (correction component). The correction function \(C()\) for the relative interaction position \(J_{x,y}\) given the relative viewpoint \(V_{x,y,z}\) and the screen offset \(o\) is defined in Equation 5.11:

\[
C(V_{x,y,z}, J_{x,y}, o) = J_{x,y} - V_{x,y}(1 \cdot o)(V_z + o)^{-1} \tag{5.11}
\]

The relative interaction position and the relative viewpoint are defined as offset between the real (absolute) interaction point \(I\) and viewpoint \(V\) by \(J_{x,y} = I_{x,y} - T_{x,y}\) and \(V_{x,y} = V_{x,y} - T_{x,y}\).

To apply the correction (given by Equation 5.11), the tracking data must be transformed into the display coordinate system in terms of rotation and translation (preprocessor component).
5.6 Tracking correction controller

(a) 1-dimensional parallax correction (schematic)  

(b) Tracking correction controller architecture

Figure 5.29: Tracking based parallax error correction

5.6.1 Sensor hardware

To track the user in front of the interactive screen, the Kinect vision sensor [5] is used. The device provides a depth map of the environment as well as a standard color image with a resolution of 640x480 px (VGA) at a refresh rate of approximately 30 Hz. Moreover, the Kinect includes an accelerometer, as well as microphones, which are not used in the tracker application.

The integrated depth sensor consists of two elements. On the one side of the Kinect, there is a laser projector that emits a pattern of infrared speckles into the room. In order to calculate the depth from this 2D information, the pattern is known at a certain reference plane. The distances between the speckles when being projected on this reference plane are stored in the memory of the Kinect. The second element is an infrared camera. If an object will change its position with regard to the camera, the distances between the speckles will change as well, which will then be used to calculate the depth information. The depth value is calculated for every pixel from this disparity between reference- and real projection. The generated depth pixel map is then send to the software framework. For further information about the mathematical model behind the depth sensor see [120].

5.6.2 Stabilized viewpoint tracker

Based on visual object detection, the OpenNI software framework, which was released by PrimeSense, supports object detection, full-body skeleton tracking and simple gesture recognition. Compared to other commercially available motion capture systems such as Vicon [26] or Qualisys [17], the Kinect does not require markers or other additional hardware to be attached to the body. Moreover, high quality tracking system are expensive and not applicable for mass products.

The orientation of the device can be detected using accelerometers, while a built-in ‘user-generator’ tracks moving objects using a motion flow detector. If an object moves in front of the camera, the user generator will assign an ID to the particular pixels of that specific object. Figure 5.30 shows a depth image of a detected user (marked as blue) object without skeleton detection.

The framework also has the possibility to track the 3-dimensional position of the major body joints like the head, torso, both shoulders, elbows, hands, hips, knees and feet (skeleton tracker). To do so, the user has to perform a specified calibration pose prior to the actual application [128]. However, this is not a natural behavior for a user. In order to avoid this initial posture calibration, a simplified viewpoint
5 Parallax error correction

Figure 5.30: Vision based user tracking (depth sensor image)

tracking method is used, which does not rely on such an initial calibration pose. Since it is only been
focused on the user’s viewpoint, the highest identified pixel in the depth map of the user generator is
used. For defining the current viewpoint, 120 [mm] in gravitational direction are subtracted from this
highest detected pixel, while the gravitational direction is deduced from the accelerometers of the Kinect.
The Kinect driver then transfers the tracking data to the user aware application. One important aspect that
has to be taken into account is the transformation from display to camera coordinates, since the tracking
system captures the viewpoint in camera coordinates, while the user space above the tabletop application
is defined in display coordinates. Therefore, the position and orientation from the camera relative to the
screen is required in order to transform and rotate the coordinates and the pixel pitch of the display to
scale the coordinates. To enable a precise tracking in display coordinates, an accurate calibration of the
camera and the display position is crucial.

An accurate tracking and a good alignment between the coordinate system of the tracking system and
the display system is crucial to precisely correct the interaction error. The depth map accuracy of the
Kinect was found to be less than ±20 mm [120]. The viewpoint position of the Kinect based tracking is
compared to a high quality reference tracking system from InterSense in terms of stability and precision.
The measurements confirm these results for tracking the user’s head based on the skeleton tracker with a
deviation of ±30 mm.

The measurements of the Kinect, and therefore the user’s position information as well, are highly noisy.
In addition, the resolution of the measurements decreases with increasing distance to the object [90]. For
these reasons, the use of filter methods is inevitable.

An exponential smoothing filter is implemented. This is an efficient and effective first order infinite
impulse response filter (IIR filter), which is defined in Equation 5.12.

\[ y(k) = \alpha \cdot y(k - 1) + (1 - \alpha) \cdot x(k) \]  

The value \( y \) is updated in time steps \( k \) as a weighted combination of the value at previous time step \( k - 1 \)
and the new measurement \( x \) at time \( k \). The weighting factor is defined by the variable \( \alpha \in [0, 1) \). In
experiments it is figured out that 0.87 is a reasonable value for \( \alpha \); providing a good trade-off between
noise suppression and time delay.

A drawback of the exponential smoothing filter is its slow step response. If a user walks from one side
of the large whiteboard to the opposite one and interacts from there again, the system experiences a step
5.6 Tracking correction controller

in the correction based on the viewpoint information during his next interaction, since the user is only tracked while he is interacting with the surface. For this case, the filtered viewpoint shall quickly adapt to the new viewpoint. Thus, the IIR filter is modified with a signal depending alpha value, as shown in Equation 5.13 and 5.14.

\[
\alpha = \Theta(\Delta_{\text{max}} - \Delta) \cdot \left( \frac{\alpha_{\text{max}} - \alpha_{\text{min}}}{1 - e^{b|\Delta|-c}} + \alpha_{\text{min}} \right)
\]

\[
\Delta = \frac{x(k) - y(k-1)}{\sigma}
\]

Figure 5.31: Viewpoint tracking filter

Where \( \Theta \) is the Heaviside function and \( \Delta \) represents the discrepancy between the current filter output and the measured viewpoint position normalized to the standard deviation of the noise signal \( \sigma \).

\( \alpha_{\text{max}}, \alpha_{\text{min}}, b, c, \) and \( \Delta_{\text{max}} \) are the filter parameters. From equation 5.13, the plot shown in Figure 5.31(a) is derived. It shows the relationship between \( \Delta \) and \( \alpha \).

If the system experiences a step input, an alpha value close to zero causes the rapid adaption of the measured, new position of the user \( x(k) \). The Heaviside function separates steps from steady input signals. If a step is given into the system which exceeds the filter parameter \( \Delta_{\text{max}} \), \( \alpha \) is set to zero. This causes an immediate step response.

If, in contrast, the user is moving slowly, \( \Delta \) is normal distributed and therefore within the range of a few standard deviations. In this case, the value of \( \alpha \) is almost one which ensures good noise suppression.

If the user is walking during the interaction, \( \Delta \) increases. Hence, a tradeoff between multiple objectives, i.e. time delay, noise suppression, and accuracy must be found. Those characteristics are defined by the filter parameters \( \alpha_{\text{min}}, b, \) and \( c \) which are shown in Figure 5.31(a). Experimental studies showed that the extended filter gives a good performance if the parameters are set to the following values:

\[
\alpha_{\text{max}} = 0.994, \alpha_{\text{min}} = 0.6, b = 2.5, c = 11, \Delta_{\text{max}} = 10
\]
5 Parallax error correction

Figure 5.31(b) shows the output of the extended and the IIR filter exemplarily. The measurement data is collected by tracking the viewpoint of a stagnant person, standing in one meter distance to the camera. Afterwards, the x-measurements were extracted and some normal distributed noise with a standard deviation of six millimeters was added to the measurement in order to test the effect of the noise suppression.

Since both filters are initialized for a initial viewpoint at the screen origin, the beginning of this set of data is equivalent to a negative step input. The nonlinear filter extension has a much faster step response compared to the IIR filter, which takes approximately 0.6 s to adapt to the new position. Furthermore, the nonlinear extension has a better noise suppression than the IIR filter, although the IIR filter provides higher accuracy.

Finally, the three-dimensional location information of the user is transformed into display coordinates and send to the correction controller.

5.6.3 Implementation

The callback in Algorithm 1 extracts the head position from user tracker and depth camera. It is triggered by the update loop of the OpenNI framework `g_Context.WaitForAndUpdateAll()`. 

```
const OFFSET = 120
viewpoint = [0,0,0]
while (g_Context.WaitForAndUpdateAll()) do
    idMap = sceneMD.Data()
    depthMap = depthMD.Data()
    gravity = getGVector()
    for pixelRow in depthMap.rows do
        for pixelColumn in depthMap.columns do
            idPixel = idMap[pixelRow][pixelColumn]
            if idPixel != 0 then
                depthPixel = depthMap[pixelRow][pixelColumn]
                top = [pixelRow, pixelColumn, depthPixel]
                viewpoint = top + gravity * OFFSET
            end if
        end for
    end for
end while
```

The algorithm is written in C as listed in Appendix B.

5.7 POMDP controller

As introduced in Section 1.2, sequential decision making is a process in which a controller at each step receives some information about the world (observation) and selects an action based on the accumulation of this information (current belief state) and a control policy. The information received is incomplete and the results of actions are uncertain.

The parallax correction controller continuously adapts the error correction to a viewpoint estimate, which
is continuously updated by an observation model and a process model. Given the actual viewpoint, which is unknown to the controller, the observation model defines the probability of the occurrence of observation for a given state. The controller receives one of the observations and calculates a probability distribution over the state space (viewpoint) applying the Bayes theorem to the conditional probability of the observation model. The observations are gathered from interaction events on (or nearby) GUI elements that have well-defined focal points (e.g., small buttons), as shown in Figure 5.22. The offset between the focal point of these widgets and the interaction point is the interaction error. Comparing several such interaction errors with a reference model of offset distributions for different viewpoints allows calculating a probability distribution over the different possible user positions relative to the target. To estimate the user’s viewpoint location for the next interaction on the interactive surface, the underlying discrete system dynamics model stochastically expresses the user’s movement within the next discrete time control step. Conceptually, each of the possible positions represents a hypothesis. Once the software component has enough evidence supporting one of these hypotheses, it adapts the assumed viewpoint position and recalibrates the correction parameter for the next interaction correspondingly.

An optimal (and non-myopic) policy for such a process is a mapping of the accumulated information to action choices that maximizes the expected value of some value (or reward)-function over time. The value-fuction models two objectives: The irritation of the user due to the changing correction and the resulting interaction error after applying the parallax correction.

Hence, correcting the parallax error based on imprecise interaction observations can be formulated as sequential decision making problem under uncertainty [147, 149, 148]. A common formalization for such problems is a POMDP. As introduced in Chapter 2, POMDPs provide a framework for non-myopic sequential decision making to control dynamic systems under uncertainty. A POMDP combines Bayesian estimation (see Section 2.5.1) and model predictive control under uncertainty. The non-myopic control policy is automatically deduced from planning out a model (see Section 2.5.2). This enables the controller to take into account the level of certainty about the current system state when selecting the appropriate control action.

Due to the fact of being defined relative to the currently assumed viewpoint, the controller’s model gets small enough to apply online planning and to be implemented into the embedded system.

5.7.1 Formal problem description

Finding the optimal control policies of POMDPs is \textit{NP-hard} [135]. Hence, expressing the given problem as a small model is necessary to apply a POMDP controller. To realistically model the problem of correcting the parallax error, the model is deduced from identifying the user behavior using interactive screens, which is described in Section 5.5. In the following chapter, the collected data is manually transferred into a small POMDP model.

The user’s viewpoint location is modeled as state of the discrete deviation space (see Figure 5.32). It is assumed that the user’s viewpoint moves along a discretized x-y plane in front of the display, since the distance to the screen is stable (see Section 5.5.1). The measurement results in Section 5.5.1 show no correlation between the horizontal and the vertical dimension of the user’s viewpoint location. Thus, the dimensions are assumed to be independent. Due to symmetry, the x (horizontal) and y (vertical) axis is identical in the model derivation and restrict the following discussion to movements in the x-axis. For simplicity reasons, an infinite screen is assumed to allow a simpler model with fine granular tracking characteristics in the majority of possible user positions. This allows for a regular model, in which the position on the discretized x-axis represents the deviation between the inferred viewpoint the controller compensates for and the actual position of the user’s viewpoint. Possible (discrete) interaction
points on the screen are realistically limited by the user’s arm length and they are distributed around his perpendicular screen position as analyzed in Section 5.5.1 (see Figure 5.15). Within this interaction zone, each potential hit point corresponds to a correction setting that is derived by the inferred viewpoint and the actual target. These basic geometrical properties in the x-z plane are shown in Figure 5.32.

In the following it is shown in detail, how the compensation problem can be modeled as a POMDP.

5.7.2 Model definition

As introduced in Section 2, a POMDP is defined as a 6 tuple. To model the parallax-correcting problem as a POMDP, this 6 tuple is defined as follows: The system’s actions are the application of different adjustments to the assumed viewpoint (and the parallax correction). Its observations are the user interactions measured by the hardware’s sensors for specific GUI elements, while (domain) states (S) represent the different possible distortion errors caused by the user’s viewpoint, which are not directly observable. The correction controller defines the states as deviation of the user’s actual viewpoint to the left (respectively right) from the assumed viewpoint [149]. The line between the actual assumed viewpoint and the target position defines the expected interaction position (see Figure 5.34). Hence, the offset between expected interaction position and target focal point defines the actual applied correction of the interaction, so that the interaction on the expected interaction position ($O_{hit}$) would be perfectly corrected to the center of the target. Interacting beside the assumed interaction point indicates a different viewpoint of the user and the controller adapts the assumed viewpoint according to the observation model. If the interaction point around a specific target is within the tolerance interval, the controller assumes that the respective target was aimed for. Therefore, it is mapped to the observation space and emits a discrete
A common widget-based graphical user interface (GUI) provides a set of elements (e.g. buttons), whose events indicate the user’s viewpoint. These so-called targets provide a reference point to measure (observe) the pointing error as the distance between target focal point and interaction (touch) point. The focal point for different targets is analyzed in Section 5.5.5.

As empirically proven in Section 5.5.3, the interaction error in principle depends geometrically on the viewpoint (see Figure 5.6). As the results in Section 5.5.3 indicate, the measurements contain uncertainty, since the ability to hit small areas on the screen also depends on the user’s dexterity. This uncertainty is described in the probabilistic observation model. Motivated by the fact that the correction can only be applied in a discrete way and the interactions are defined with respect to the discrete display coordinates, the model is formulated as discrete probability distribution. It expresses the correlation between the actual viewpoint (state) and the pointing accuracy on the screen (given as observation). Formally, it represents the likelihood of the agent receiving a particular observation given that the system changed to a particular state under a particular controller decision (parallax correction change). It is modeled as conditional probability distribution over the viewpoint locations (state space) \( S \) relative to the target location for a given interaction error \( o \) and correction action \( a \) as \( P(S|o,a) \). However, the correction action \( a \) influences the observation model being applied to the interaction, which is done before the controller observes the interaction. This allows to simplify the observation model expressing \( P(S|o) \) without loosing information. Since the correction was already applied to the interaction which gave the observation, the state space indicates the offset between the assumed and the actual user viewpoint position with respect to the target. Intuitively, a low error observation indicates the user’s actual position close to the estimate.

Since the parallax correction is done for the future interaction, the controller considers the process model in updating the belief state before requesting the optimal action, which is selected by the controller according to the control policy to align the assumed and the real viewpoint. The applied action shifts the assumed viewpoint and the corresponding applied correction parameter for the next interaction. Accordingly, the transition model describes the resulting distortion error after applying a correction action with respect to the current distortion error and the user’s behavior in terms of movement. It
is based on measuring the user's movement in front of an interactive screen (see Section 5.5.2) and combines the change of applied correction (action) with the system dynamics.

Finally, the reward model $R(s, a)$ defines rewards for adjustments depending on the actual distortion with respect to multiple objectives: The user irritation due to the changing correction and the resulting interaction error. The state represents the error, hence, the state costs express the resulting interaction error. And the action costs represent the user irritation.

The controller is initialized with a certain assumed viewpoint, e.g. the center of the screen. Any interaction is corrected according to this viewpoint as shown in Figure 5.33. The belief state represents the probability of the offset between the actual and the assumed user’s position. It is initialized as uniform distribution over the state space, which represents no knowledge about the system. The control loop rate is 1 Hz, which corresponds to the average interaction rate from the user study described in Section 5.5. The belief state is continuously updated with the probability distribution from the observation model and the transition model.

The following section describes the POMDP model deduction from empirical data.

### 5.7.3 Model deduction from user identification

A POMDP model consists of the discrete state, observation and actions space, the system dynamics in terms of the transition model, the observation model and the reward model. The following section presents a POMDP model for the parallax error problem, following the process of POMDP deduction from measurement data (see Section 4.2).

#### Model Topology

As stated in Section 5.3, the parallax error stems from the user’s viewpoint position. To correct the parallax error, the proposed POMDP controller estimates the user’s viewpoint from the sequence of past interaction errors.

The computational effort of deducing the optimal policy decreases with the size of the model, i.e. the number of states, observations and actions. Hence, the model should be small to be feasible.

In general, the state space expresses the user’s viewpoint, which is the 3-dimensional position in front of the interactive screen. In the following, the orientation of the state space coordination is aligned to the screen coordinates (x horizontal, y vertical) and z as distance to the screen.

The space is physically limited by the screen size and the user’s arm length, since considering the viewpoint does only make sense if the user can actually interact with the system. The average arm length of men (women) is 787 (762) mm for the 95. percentile [73]. Hence, the z dimension is limited by 0 and 787, the maximal arm length for 95% of all humans. The horizontal and vertical dimensions are limited by the screen size plus 787 mm.

The size of a state space, which describes the position in three degrees of freedom in mm for a large interactive screen of the size 1000 × 500 mm, is $(787 + 1000 + 787) \times (787 + 500 + 787) \times 787 = 4201380612$. Even for a small screen of the size 40 × 40 mm, the number of states is 2050131852. Decreasing the resolution from mm to cm decreases the number of state to 4113408 and 1996800, which is still too large to compute a control policy within reasonable time. A smaller resolution would lead to a limited and inaccurate state estimation for correcting the parallax error within pixel on the interactive screen.
5.7 POMDP controller

Separating the vertical and horizontal dimension significantly reduces the state space by one dimension. Two controllers separately control the parallax correction, each for one dimension, under the assumption that the horizontal and vertical component are independent, which is intuitively reasonable for the user's viewpoint.

Reducing the state space to one dimension is preferable to further reduce the model complexity. As shown in Figure 5.35(b), the independence does not hold between the z- and x-, as well as the z- and the y-dimension. An interaction event defines the line that goes through the interaction position and the target position. Assuming that the user interacts in line of sight, this observation is an evidence for the user's viewpoint position, shown as gray segments. Unfortunately, the x (and y) component depends on the distance to the screen in z direction.

To map an observation onto a one-dimensional space, the stability of the distance between user and display is investigated in detail. Investigating the user's viewpoint position while using a large interactive screen, as described in detail in Table 5.2(a), yielded that the distance between user and screen is stable. The 95 quantile around the mean value of 577.2 mm is given by the interval [491.2, 662.82].

Based on that, the state space can be reduced by the z-dimension as follows. Given a reference line $R_z$ (at the mean value), the probability distribution that describes the user’s distance to the screen $P(V_z - R_z)$, is mapped to the Reduced viewpoint space in x (y) dimension with respect to the observation as shown in Figure 5.35(a). The observation beam indicates every point on the line between points A and C. With respect to the probability distribution $P(V_z - R_z)$ it is mapped geometrically along the mirror plane $AB$ to the Reduced viewpoint space, i.e. $A$ is mapped to $B$, $C$ is mapped to $D$, $E$ stays $E$. The real points $A, C, E$ are mapped to virtual points $B, D, E$ on the reduced space.

Hence, the information that describes the z component of the viewpoint is moved to the observation model, which describes the correlation between interaction (observation) and the viewpoint (state). In order to set up the model realistically, the models are deduced from studying the user behavior on interactive screens.

**Model deduction from measurement data**

Based on the model topology, the observation model and the transition model are deduced from empirically studying the user’s behavior interacting on an interactive screen (see Section 5.5). Significant
distinguishably of the viewpoints (states) for given observations is necessary for a good measurement model. The distinguishably of the empirical probability distribution can be established by adapting the underlying interval of observations and states. Hence, the distinguishably of the empirical probability distributions influences the observation and the state space discretization. The state does not represent the current viewpoint of the user, but the deviation from the currently assumed viewpoint, the actions represent shifting the applied error correction instead of setting the correction. The effect of the action is handled by the controller i/o interface, since the interaction observation is passed to the controller after the error correction is applied. The system dynamics in terms of user movement defines the transition model. It is deduced from tracking the user’s viewpoint in front of an interactive screen (see Section 5.5.1). Although the controller state does not model the user’s current viewpoint location, but its deviation from the assumed viewpoint, the dynamics in terms of movement are the same. The same holds true for the observation model: It indicates the current viewpoint, which is used by the controller to update its belief state.

Since the measurements are given in mm, the model unit is mm.

State space, observation space and observation model

The observation index is deduced from a one-dimensional discretization of the measures. The interaction error is deduced as offset between the corrected actual interaction position on the interactive screen and the target’s focal point. Since the screen locations are given in display coordinates, the upper bound of the model discretization is given by the display resolution.

The observation space discretization is given by the observation model deduction from empirical data. The observation model describes the correlation between states and observation. To distinguish the state based on observations, a strong correlation between the observation and the corresponding state is needed. The discretization of the underlying domains is defined regarding this distinguishably of states by given observations.

The POMDP state space represents the deviation from the assumed viewpoint and the real viewpoint of the user. It is an artificial internal representation of the controller. The states are identified by index. The state space discretization is defined according to the observation model as described below.

From the controller’s perspective, an observation is expressed as index over the set of observations. Nevertheless, the observation is given by a measurement on the interactive system, in particular the interaction error (see Definition 10). To set up a regular model reducing the number of observations and states correspondingly, the observation model represents the uncertainty in the correlation between the interaction error (observation) and offset from the assumed viewpoint (state); in other words - the uncertainty of the sensor value describing the real system state. This uncertainty is formally expressed as observation model $\Theta$. Formally, the observation model is given by $P(s|o) = \frac{P(o|s) \cdot P(s)}{\sum_s P(o|s) \cdot P(s)}$ for the observation $o$ and the state $s$. However, a good observation model maps observations with low uncertainty to states to enable a high confidence of the controller’s belief state within the Bayesian filter update step. Hence, the state space and observation discretization is selected accordingly.

The state space prior $P(s)$ models the independent probability distribution of the viewpoint relative to the interaction point w.r.t. the state space discretization. It can be immediately deduced from measuring the viewpoint while the user interacts on the system, as described in Section 5.5.1. The second aspect of the observation model is $P(o|s)$. It is the correlation between the interaction error and the viewpoint relative to targets (see Figure 5.34). The correlation considers two aspects. The effect of the area target’s shape and the interaction error that stems from the user’s dexterity. The effect of the target shape in terms of size and content is identified in Section 5.5.5. The results show that the target content has a
5.7 POMDP controller

The state space is discretized such that the states are distinguishable by observations. There are five intervals representing the five viewpoint deviations ($s_0 \ldots s_4$) for left, left, center, right and right from the target’s location ($[-\infty, -0.45, -0.15, 0.15, 0.45, +\infty]$). Since the correction is already applied to the observation and the correction depends on the currently assumed viewpoint, the intervals also represent the deviation from the assumed viewpoint.

The measurement results in terms of interaction error and relative viewpoint from Section 5.5 are normalized by the distance to the screen.

The observation space is discretized accordingly. The box plots in Figure 5.36(a) show that the areas between the first and the third quartile do not overlap for all states except for far right, since the measurement data do not provide sufficient data for the interval $[0.45, +\infty)$ due to the few left handed participants of the study. Figure 5.36(b) and 5.36(c) show the observation density for the states slightly left and slightly right in detail. The shape of the function is clearly not Gaussian distributed.

Due to the significant difference of measured interaction observations for given discrete states, the interaction error can be utilized to indicate the viewpoint. In the following, it is shown how the discrete observation model is deduced from the measurement data.

Since the interaction error is measured after the correction was applied, the observation model does not depend on the action, and the findings from Section 5.5.3 are translated into conditional probabilities of the form $P(S|O = o)$, such that the i-th row of the observation matrix represents the probability distribution over the system states given the i-th observation. The resulting observation model
5 Parallax error correction

\[ O(s, a, o) = P(s|o) = \frac{P(o|s)P(s)}{\sum_s P(o|s)P(s)} \]

is given in Equation 5.16

\[ \Theta_{i,j} = P(s = s_j|o = o_i) \]

\[ \Theta = \begin{pmatrix}
1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.64878052 & 0.35121948 & 0.0 & 0.0 & 0.0 \\
0.28358209 & 0.35820895 & 0.35820896 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.64285715 & 0.32142856 & 0.03571429 \\
0.0 & 0.0 & 0.85714286 & 0.14285714 & 0.0
\end{pmatrix} \]

(5.16)

It is deduced from \( P(o|s) \) and \( P(s) \) given by Equation 5.17 and 5.18.

\[ \bar{\Theta}_{i,j} = P(o = o_j|s = s_i) \]

\[ \bar{\Theta} = \begin{pmatrix}
1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.7 & 0.2 & 0.0 & 0.0 & 0.0 \\
0.3 & 0.2 & 0.4 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.4 & 0.4 & 0.1 \\
0.0 & 0.0 & 0.75 & 0.25 & 0.0
\end{pmatrix} \]

(5.17)

and the prior state probability

\[ P(s = s_i) = \left( 0.22093023 \ 0.41860465 \ 0.20930233 \ 0.10465116 \ 0.04651163 \right) \]

(5.18)

The knowledge of the prior state distribution shifts the correlation model towards the second state which has the highest probability.

**Action space and transition model** The actions enable the controller to transfer the system from one state to another with respect to uncertainty. Since the states represent the deviation between the assumed viewpoint and the real viewpoint, the controller transfers the system state by changing the correction parameter and shifting the assumed viewpoint to compensate the parallax error. The actions are defined with respect to the state space discretization as follows. Three actions to transfer the system between the five system states: shift left, no shift, and shift left. Shifting means changing the assumed viewpoint and the corresponding correction settings. Since the state space is discretized into equally sized intervals, the three actions are sufficient to control the given problem. The correction value of the actions is defined according to the normalized state space discretization. Since the interval size is 0.3, the value of shift left is \(-0.3\) and for shift right it is \(+0.3\). The index of \( a \) is set to \(-1, 0, 1\).
The transition model expresses probabilistically how the system state evolves within one discrete time step under the effect of actions. Since the system state models the position of the user’s viewpoint, the transition model expresses the movement: It expresses the probability of the next viewpoint position relative to the last one. The currently applied correction parameter offset corresponds on the current assumed viewpoint. Since the system state expresses the deviation between the assumed and the real viewpoint, the correction action in terms of changing the applied correction parameter, also shifts the current assumed viewpoint.

Formally, the time-invariant transition model is given as $P(s'|s,a)$. The user’s movement is assumed to be independent from the applied correction action modeled as new location $s'$ given the current location $s : P(s'|s)$ within the control loop time-step. Since the state represents the offset between the actual and the assumed location of the user’s viewpoint, the action $a$, in terms of changing the parallax correction offset, influences the transition model linearly. Shifting the assumed viewpoint to the left corresponds to increasing the interaction correction (negative action value), which also shifts the transition matrix cells.

$$T(s', a, s) = P(s'|s, a) = P(s' = j|s = i, a = \bar{a}) = T_{i,j}^a \quad (5.19)$$

The transition model $T$ is set up with respect to the state space discretization. The transition probability $P(s'|s)$ is deduced from measuring the user’s movement as discussed in Section 5.5.2. It is for an arbitrary $k$ given by:

$$
\begin{pmatrix}
P(s' = s_{k-2}|s = s_k) \\
P(s' = s_{k-1}|s = s_k) \\
P(s' = s_k|s = s_k) \\
P(s' = s_{k+1}|s = s_k) \\
P(s' = s_{k+2}|s = s_k)
\end{pmatrix} =
\begin{pmatrix}
0.01339286 \\
0.16071429 \\
0.67410714 \\
0.14285714 \\
0.00892857
\end{pmatrix} \quad (5.20)
$$

This movement w.r.t. a previous position $k$ is transferred into the transition matrix for given previous positions $s_i$ as follows. The diagonal elements represent the probability of no movement given by $P(s' = s_k|s = s_k)$. The elements on the left (right) secondary diagonal a one-step movement to the left (right) given by $P(s' = s_{k+1}|s = s_k)$. Since the border states $s_0$ and $s_{|S|}$ cover the state space up to infinity, the probability mass of all state ‘outside’ the matrix is accumulated, accordingly. States that are not covered by the movement transitions in Equation 5.20 are set to zero by definition.
The resulting matrix of the user’s movement in horizontal dimension is given by Equations 5.22, 5.21 and 5.23. The vector of Equation 5.20 is shown completely at the center row of the matrix, since the corresponding source state is the center state $s_2$.

$$T^{a=1} = \begin{pmatrix}
0.17410715 & 0.67410714 & 0.14285714 & 0.00892857 & 0.0 \\
0.01339286 & 0.16071429 & 0.67410714 & 0.14285714 & 0.00892857 \\
0.0 & 0.01339286 & 0.16071429 & 0.67410714 & 0.15178571 \\
0.0 & 0.0 & 0.01339286 & 0.16071429 & 0.82589285 \\
0.0 & 0.0 & 0.0 & 0.01339286 & 0.98660714
\end{pmatrix}$$

(5.21)

$$T^{a=0} = \begin{pmatrix}
0.84821429 & 0.14285714 & 0.00892857 & 0.0 & 0.0 \\
0.17410715 & 0.67410714 & 0.14285714 & 0.00892857 & 0.0 \\
0.01339286 & 0.16071429 & 0.67410714 & 0.14285714 & 0.00892857 \\
0.0 & 0.01339286 & 0.16071429 & 0.67410714 & 0.15178571 \\
0.0 & 0.0 & 0.01339286 & 0.16071429 & 0.82589285
\end{pmatrix}$$

(5.22)

$$T^{a=-1} = \begin{pmatrix}
0.99107143 & 0.00892857 & 0.0 & 0.0 & 0.0 \\
0.84821429 & 0.14285714 & 0.00892857 & 0.0 & 0.0 \\
0.17410715 & 0.67410714 & 0.14285714 & 0.00892857 & 0.0 \\
0.01339286 & 0.16071429 & 0.67410714 & 0.14285714 & 0.00892857 \\
0.0 & 0.01339286 & 0.16071429 & 0.67410714 & 0.15178571
\end{pmatrix}$$

(5.23)

**Reward model**  The reward model describes the objective of the controller. Selecting the control actions optimally with respect to the objective function leads to optimally control the system. The reward model describes multiple objectives.

The first aspect is the deviation between the assumed viewpoint and the actual viewpoint, which is represented as state. The center state represents the overlap, which is the goal of the controller. The state left (right) represents a low deviation, which is bad in the sense of optimal control, and the states far left (far right) are even worse, due to a high deviation between the controller assumption and the real user viewpoint. This is expressed as state costs (negative rewards), correspondingly.

The second aspect of the reward function is the user irritation which stems from changing the screen calibration. It is assumed, that the user will also try to correct the interaction error by extrapolating
the error for future interactions based on his experienced interaction errors. Hence, a changing parallax correction might lead to irritations. The user irritation is expressed as transition costs. Independent of the state, it depends only on the fact that a correction change is applied. Hence, ‘no shift’ costs no transition, except for the state costs which are treaded separately, but *shift left* and *shift right* cause costs, which are equal due to symmetry.

Although the relation between the costs can be expressed intuitively, neither the state costs nor the action costs can be deduced from measurements realistically. However, the relation between the costs effects the control policy (see Section 2.5.5) which is shown in Figure 2.9. The cost relation effects the level of aggressiveness for uncertain belief states. If the MDP policy dominates with a high reward, it also dominates the region closer to the uniform belief due to the gradient of the linear $\alpha$ vector. If the domination is based on a low advance, the controller might prefer actions that do have an $\alpha$-vector with a low gradient (equal MDP values) in case of high uncertainty.

The controller will act on a progressive scale, if the transition costs are low and the state costs high. Hence, it is reasonable to increase the state cost in a higher order than linear to make the controller compensate the bigger deviation of the assumed viewpoint more aggressively than smaller ones.

If, however, the cost of applying an action is bigger than the state penalties (negative rewards), the controller won’t act at all. Hence, the state costs shall be greater than the action costs.

**Input/output interface**

In Section 5.7.3 is described how the POMDP model is defined. A POMDP model probabilistically defines the system dynamics and measurement model, which can be deduced from system identification. The remaining model elements states, observations and actions are defined symbolically and simply identified by index. Since observations and action define the interface between controller and real world, the symbols must be mapped to real world values in terms of measurement values of sensors and set points of actuators.

The POMDP model is defined based on observations distinguishing the state, which declares the observation and state space discretization $[-\infty, -0.45, -0.15, 0.15, 0.45, +\infty]$. Since the space is normalized by the z-distance, the measured interaction error value is normalized by pixel pitch of the display and the offset between interaction plane and display plane before it is mapped to an observation index.

The correction action is mapped accordingly. It shifts the assumed viewpoint from on state to a neighboring state. Since the normalized interval size is 0.3, the corresponding action step size is either +0.3, 0, or -0.3. The actuator value is also mapped from the normalized space to the display plane multiplied with the screen offset.

**5.7.4 Controller architecture**

The parallax corrector consists of several components as shown in Figure 5.38. The correction controller is hooked up between the pointing device and the software application running on the interactive system. As described in detail in Appendix B.2.1, the pointing device emits interaction events which carry the position in display coordinates. The calibration component of the correction controller interrupts and changes the interaction event according to the correction parameter before it forwards the event to the corresponding software application. The correction parameters are deduced from the estimate of the user’s viewpoint. They differ depending on the interaction position as introduced in Section 5.7.1 and shown in Figure 5.33.
The correction parameters are continuously adapted by the decision maker's actions, which are defined in Section 5.7.3. The core of the correction controller is the decision maker. It keeps a belief state of the current deviation between actual and assumed viewpoint of the user. The belief represents the current knowledge of the controller about the system state and covers uncertainty. Based on the current belief, the optimal control is requested from the policy which is calculated by the planner, as discussed in Section 5.7.5. The discrete observation represents the user's interaction error after the controller's correction is applied and is deduced according to the I/O-interface described in Section 5.7.3. The interaction error is calculated as offset between the interaction position and the closest target (widget), which is gathered from the software application, by the preprocessor. The implementation of the preprocessor and calibration component is listed in Appendix B.2.

To deal with changing users, the null-observation is introduced in Section 4.4. If no observation takes place within the control loop step, it transfers the controllers belief state stepwise to the uniform belief state (see Section 4.4), which represents having no knowledge about the current state of the system. The effect of the belief-state transformation is that the controller reacts with restraint to upcoming observations. According to the method described in Section 4.3, the current estimate of the user's viewpoint, which is adapted by the controller's actions, is moved step-wise towards the center of the screen. This is reasonable according to the prior distribution of the user's viewpoint. The user's viewpoint is initially mostly located at the center of the screen.

### 5.7.5 Policy deduction

The policy deduction is done by the planner component, introduced in Section 5.7.4. As introduced in Section 2.5.4, POMDP planning simulates the system behavior based on a model to deduce the optimal control policy from investigating the long-term reward of actions.

The MDP and POMDP policy for the parallax error problem is deduced by the POMDP solver software emphpomdp-solve (Version 5.3) written by Cassandra [13]. See Section 3.2.2 for details.
5.8 Controller evaluation

To use the solver, the model is expressed in Cassandra’s POMDP (see Appendix B.2.3). Since the small model is not adapted online, the policy is calculated offline and represented as α vectors in a text based file. This policy-file and the model-description is loaded as policy by the POMDP controller software, which is written in Python [15].

5.7.6 Conclusion

In the previous chapter, a POMDP controller is designed to correct the parallax error on interactive screens. First, the given problem is mapped onto the methodology of POMDPs. The formal problem description defines observations, states and actions, and lists assumptions and simplifications to reduce the model size, such that the resulting POMDP is small. Then, the model is defined in detail. The topology in terms of the state space, the observation space and the action space is defined and the observation-model and the transition-model are deduced from the results of identifying the user behavior. Next, the reward model is filled with reasonable values. The I/O-Interface defines the mapping between symbolic actions and observations and their corresponding real-world values. The controller architecture defines the integration of the controller into the software framework of the interactive system and refers to the implementation of the prototype. Finally, the policy deduction is described. The solver deduces the control policy for a given POMDP definition.

5.8 Controller evaluation

In order to evaluate the correction behavior of the parallax error controller, the following section shows the correction behavior of the controller for simulated interactions. The controllers initial response characteristic, the correction convergence in terms of gradient and convergence value are analyzed.

The measures contain the raw interaction error and the error after the correction has been applied over time. It is shown how fast the correction is applied and how the controller reacts on changing observations. The response characteristic in terms of initial correction behavior is evaluated. It describes how fast the controller is convinced about the system being in a certain state by the sequence of observations to apply a correction action. Analyzing the correction of a static interaction position also shows the gradient of the interaction correction and the convergence of the correction. Due to the model structure, it is assumed that the correction will be changed linearly and the resulting interaction error will converge against zero.

The only free parameter to tune the controller’s reaction behavior is the reward model which is not deduced from measurement data. To evaluate the influence of the reward model, three different controllers are evaluated, as described below in two scenarios.

First, a static interaction position is sampled against one target. This target represents arbitrary targets on the interactive screen, since the parallax correction controller evaluates the error after the correction is applied based on the current estimate. The second scenario simulates inconsistent observations. An oscillating interaction position is sampled to show the controller’s behavior under changing measures. It shows the robustness of the correction convergence and needed the level of certainty to apply an action.
5 Parallax error correction

5.8.1 Setup

The simulation samples a sequence of interaction errors which are sent to the controller. The controller transforms the interaction error to discrete observations, updates its internal belief state and queries the policy for the correction action according to the control loop. Eventually, the controller maps the action index to a real correction value which is returned to the simulation. The resulting interaction error is measured before and after the controller’s correction is applied.

The controller is mainly deduced from measurement data. Solely the reward model is defined manually. It is deduced from the ratio between state costs and action costs. The state cost penalize the system being in the wrong state, which means that the estimate and the deduced parallax correction is wrong. On the other hand, the action costs penalized the controller to apply an action, which causes the viewpoint estimate to change but irritates the user, since the parallax correction changes accordingly. Three different controllers with state/action cost ratios of $2 : 1$, $5 : 1$, and $11 : 1$ are evaluated.

The parallax offset is set to 10 mm. The observation space of five observations is defined by the normalized intervals $(-\infty, -0.45], (-0.45, -0.15], (-0.15, +0.15], (+0.15, +0.45]$, and $(+0.45, +\infty)$. It is mapped onto the display space to the intervals $(-\infty, -18], (-18, -6], (-6, +6], (+6, +18], (+18, +\infty)$ px based on a pixel pitch of 0.255 mm. The three actions $([-0.3, 0, 0.3])$ shift the viewpoint estimation by -173, 0, and +173 mm in the viewpoint plane with a distance of 577 mm to the display. Hence, the correction shifts the interaction correction either by zero or $\pm 12$ px. The frequency of interaction is set to 1 Hz. The initial viewpoint estimate is set perpendicular to the target position.

The static interaction is evaluated over 20 time steps with a raw interaction error of $\pm 40 (+40)$ px ($\pm 10$ mm). This causes an offset of 5 mm between the interaction position and the target detection for a target of the size 10x10 mm. Hence, the interaction position should be shifted by 5 mm to be interpreted correctly. Since the controller observes the corrected interaction error, it initially represents $s_0(s_4)$ and converges against $s_3$.

The oscillating interaction error changes every time step between the value pairs $(-40, 0), (40, 0)$ which (initially) represent state $s_0$ and $s_3$ without any applied correction. Additionally to this large amplitude of 20 px, the sequence of observations is analyzed for all pairs of direct neighbors $(o_i, o_{i+1})$ $i \in \{0, \ldots, 3\}$ by an oscillation amplitude of 6 px.

5.8.2 Results

Figure 5.39 shows the correction of a static interaction offset of $\pm 40$ px ($\pm 10$ mm). As shown in Figure 5.39, the correction for a target of the size 10x10 mm is reached after 8 respectively 10 interactions by the controller. The controller with the higher state/action cost ratio reacts initially faster, which is reasonable, since the state is penalized stronger and the controller tries faster to leave the wrong state applying actions. Figure 5.39(c) shows that the controller acts symmetrically. The positive interaction error is similarly corrected to the negative errors, shown in Figure 5.39(a) and 5.39(b). In any case, the corrected interaction error converges against the value of -12 px.

Figure 5.40 shows the correction for an oscillating interaction error with a small amplitude. The observations alternate between adjacent observations. The correction of the correction controller with the cost ratio of 2:1 (5:1) is shown in Figure 5.40(a) (5.40(b)). It shows that the controller corrects the interaction error towards zero, although the observations oscillate. The controller converges after alternately getting observations $o_2$ and $o_3$. The lower state penalty makes the controller react slower onto the interaction error, see Figure 5.40(a) and 5.40(b). The correction of measured observations representing the equiva-
5.8 Controller evaluation

![Graphs showing error vs interaction index for different cost ratios](image)

Figure 5.39: Correction of a static interaction error

![Graphs showing error vs interaction index for different cost ratios](image)

Figure 5.40: Correction of oscillating interaction error with small amplitude

lent on the right hand side of the model are shown in Figure 5.40(c). On the right side, the controller’s behavior does not differ depending on the reward model.

Figure 5.41 shows the correction for an oscillating interaction error with a large amplitude of 20 [px]. The observations alternate between the observation pairs \((o_i, o_{i+2})\) \(i \in \{0, 1, 2\}\). Although the observation alternates between a high error and no error, the positive interaction error in Figure 5.41(a) is shifted. After 20 interactions, the correction converges at a value of \(-18\) px. The equivalent negative error does converge similarly for a cost ratio of 5:1 and 11:1 as shown in 5.41(c). However, it does not converge to the same value by applying no correction for the same scenario with a cost ratio of 2:1 as shown in Figure 5.41(b).

5.8.3 Conclusion

Analyzing the static interaction error shows that the controller reacts correctly in any state. It corrects the interaction towards the target; to the right getting observations \(o_0\) or \(o_2\), to the left getting observations \(o_3\) or \(o_4\), and it does not change the correction once the observation indicates the system being in the desired state \(s_3\). The corrected interaction error converges against the value of -12 px, due to the observation model. The reason for this is that getting observation \(o_2\) gives a high evidence for the system being in the desired state \(s_3\), which causes the controller not to apply any correction action. The level of progressivity can be controlled by the reward model. If the state/action penalty ratio is increased, the controller reacts more aggressive changing the viewpoint estimate. The correction converges linearly, which is reasonable due to the structure of the underlying model. The controller can either apply and
correction action to switch the system between adjacent states or let the system stay in the current state by not changing the viewpoint estimate. Since, the controller can not skip states, the correction change is linear.

Analyzing oscillating interaction errors shows that the controller is also robust against changing observations. Although getting alternating adjacent observations, the controller corrects the interaction error. Since the amplitude of 6 px lies within the range of observation $\omega_2$ and $\omega_3$, which is $(-18, +6]$, and both observations indicate the desired state $s_3$, the controller does not change the correction while getting one of these measurements. This holds true for positive and negative interaction errors. Getting large alternating observations makes the controller to react more cautious. Since the sequence of observations do not clearly indicate the system to be in a certain state, the controller applies less correction actions.

### 5.9 Conclusion

The question, which are formulated in Section 5.4.10, are answered in the following.

Studying the parallax error shows that the interaction error, which stems from the parallax offset between the interaction plane and the display plane, can not be corrected with static calibration, since the error stems from the user’s viewpoint and the user moves in front of the screen while interacting on the digital surface.

The development of the tracking controller in Section 5.6 shows that it is possible to directly capture the user viewpoint in order to correct the parallax error.

It is also possible to estimate the user viewpoint without additional hardware with the POMDP estimation controller presented in Section 5.7. The controller utilizes the previous interaction of the user to update the viewpoint estimate. The estimation controller takes into account the user irritation that stems from changing the estimation. Moreover, it deals with changing user using the principle of null observations to develop its belief state as proposed to model oblivion in Section 4.4. To model the problem realistically, the POMDP controller model is deduced from identifying the user behavior in Section 5.5.

#### 5.9.1 Summary

This chapter described the research on correcting the parallax error. After introducing the parallax error, it demonstrates the effect of the parallax error on the interaction accuracy by measurements.
Two parallax error correction systems are presented. One, that direly tracks the user’s viewpoint, and another one, that estimates the viewpoint from preceding interactions on the display. The benefit of the estimating system is clearly the costs of the system, since no additional hardware - like a tracking system - is needed. The problem of dealing with changing users is solved by introducing Null-observations, which let the viewpoint estimate converge against the center of the screen and the belief state against the uniform belief when no interaction takes place.

To deduce a realistic model from measurement data, the user behavior identification shows how the user behaves in front of large interactive screens. The user’s viewpoint is investigated in terms of position and movement, which is later transferred into the system dynamics model of the controller. The correlation between the interaction error and the user’s viewpoint is investigated and transferred into the measurement (or observation) of the controller. Moreover, it is shown, that the interaction precision does not increase using a pen TUI instead the bare finger, and that the precision increases linearly with the size of the target area. To provide a single target point, the focal points of area widgets are investigated. It is shown, that the user mostly aims to hit the geometrical center of targets, which differs from the optical center given by the content of the target.

The measurement data is transferred into a POMDP model. It describes the system dynamics as transition model and the correlation between measurements and system states as observation model. Both are based on the state, action and observation space definition. The spaces are deduced from measurement data by discretization such that the transition model and observation model are meaningful, i.e. defining a state does not make sense, if no measurement data of it’s dynamics is available.

### 5.9.2 Discussion

The main issue of modeling POMDP is the close coupling of the model elements (spaces and models). If one element is changed, the others are effected. However, deducing model aspects from measurements does not allow tweaking the model. This limitation might lead to inconsonance, but assures that the model is valid. Investigating the behavior of the controller depends on all model elements. Hence, evaluating parts of the model is hard to achieve.

POMDPs formulate discrete time and discrete space (state, action, observation) problems which require predictive control under uncertainty in terms of measurements and control effects.

A core aspect of the state, action and observation space is the discretization and dimensioning. Continuous problems require a high resolution which causes large space. Large spaces, on the other hand, increase the model complexity. If, however, a space element is defined, the corresponding aspect in the transition and observation model must be covered by the measurements, which might not be clear already before the measurements are performed. Hence, the discretization is deduced from the measurements.

The POMDP parallax error correction controller controls the estimate of the user’s viewpoint position, which defines the interaction correction depending on the interaction position on the screen.

Since the viewpoint must be estimated with high accuracy and a POMDP model, contrastingly, should be small, the POMDP model is defines relatively. More precisely, the state space defines the offset between the actual viewpoint, which correlates with the observed interaction error after the correction was applied, and the assumed (estimated) viewpoint, from which the applied correction is deduced. The controller’s actions are the adjustment of the viewpoint estimate. Hence, the problem can be formulated as sequential decision problem under uncertainty and apply POMDPs. Controlling the estimate combines uncertainty with an optimality criteria. Hence, it enables to control the estimate carefully. The level of aggressiveness is driven by the reward model, since the observation model and the transition model are...
Parallax error correction

deduced from measurement data. However, the reward model defines the relation between two optimality aspects: User irritation by changing the viewpoint estimate (and resulting error correction) and the error of the estimate. The ratio between the two aspects drives the controller’s level of aggressiveness. It is the only parameter which is defined manually.

Alternatively to the POMDP controller, common estimators might be applicable to solve the parallax error problem. A Kalman filter is not applicable to estimate the user’s viewpoint, since the measured probability distributions are not Gaussian. Hence, it might be possible using a Particle filter. However, although model-based estimators deal with uncertainty and also use a process model, they do not consider any optimality criteria to change the estimate. In contrast, the POMDP controller considers the multiple goals of a good estimate and changing the estimate. The level of aggressiveness to change the estimate depends on the reward level, more precise the ratio between state and action costs. Hence, the agility of the estimator can be adjusted by the state/action ratio.
Case Study: Strategic power plant production control

This chapter shows the application of intelligent control in the field of power production in a waste to energy (WtE) plant. In contrast to common combustion plants, the heating value of waste is uncertain. Hence, the energy production contains uncertainty. Due to the thermodynamic behavior of the plant dynamics, strategic control under uncertainty is applied to control the power plant in order to optimally meet the production demand. This is, for example, changing the production load to stabilize the electricity grid.

6.1 Introduction and motivation

As one of the basic inputs, energy is one of the most important factors of economical and industrial progression. Due to an increasing concern about efficient usage, effective transport and demand driven production, intelligent systems can help to optimize the energy management to reduce overproduction and protect the environment.

The liberalization of the electricity market in Europe and the emerging renewable electricity producers present challenges and opportunities, such as quickly reacting to changing supply and demand; as well as to existing players, like thermal plants, in the power generation industry. To optimally deal with the new situation, i.e. dynamic market rewards, new technologies, such as intelligent controllers, must be applied to energy plant control in order to increase competitiveness at the liberalized market.

As introduced in Section 1, an intelligent controller (or agent) implements capabilities such as perceive, interpret, learn, reason, plan and decide. Systems that have such capabilities mostly use techniques that are spawned in the research field of Artificial intelligence (such as expert systems, neuronal networks, genetic algorithms, fuzzy logic, planning under uncertainty). In the energy sector, intelligent computer applications became widespread in the 1990 [145] and are still an ongoing research topic [157], i.e. the integration of fluctuating power from renewable energy into the electricity grid [138] and load forecasting of electricity consumption [59] due to the increasing demand.

In general, the energy sector is composed of energy production by power plants or reservoirs, energy distribution (electric grid) and the energy consumption of industry as well as private and public institutions. The following chapter focuses on energy production, in particular energy production of waste incineration plants.
6.1.1 Waste incineration plants

The primary function of power plants is converting the energy locked in some form of fuel into power (electricity or heat). In combined power plants, the output energy is gained from an intermediate water-steam cycle by steam turbines and heat exchangers as well as (flue) gas turbines. The cycle is fed by heating energy. The heat used for steam production is obtained from burning fossil fuel (oil, gas or coal) in a conventional thermal power plant. Whereas in a nuclear power plant, it is derived from the radioactive decay. In contrast, renewable energy plants produce electric energy from solar radiation or wind. The fuel of a waste incineration plant is domestic or industrial waste. Due to the nature of the fuel material, the pollution of the plant is critical and strictly regulated. The nature of the flue gas does, however, not allow to apply gas turbines, as usual in combined power plants.

Compared to gas or coal, waste has inhomogeneous composition, volume and caloric value which makes the combustion process unstable, hard to control and predict and its energy output uncertain. Hence, in the early days, the purpose of operating a waste incineration plant was only reducing the volume of the waste. Today, disposing waste underlies strong regulations, such as little organic carbon as possible [75].

The heating value of waste increases. Today, waste is high caloric (mostly plastics) and the combustion process is stabilized. Hence, common WtE plants feed heat energy into the local district heating network receptively process steam to a nearby industry. Additionally, excess is transformed into electric energy by a turbine and a generator as part of the water steam cycle and injected into the electric grid.

Commonly, the WtE plant is driven by the public-sector and is, usually, bound to long-term contracts with local energy consumers. Due to the liberalization of the energy market, however, supply and demand as well as the energy price got flexible. Thus, demand driven energy production gets attractive for waste to energy plants.

6.1.2 Energy market

The electric grid connects energy production plants and consumers. It must be managed such that the available energy generated by the plants meets the consumer’s demand optimally in economic manner [134] with respect to load, voltage and frequency. The history of energy production in the European Union raised from \(2.8 \cdot 10^9\) in 2004 up to \(3.3 \cdot 10^6\) MWh in 2012 [4]. In Switzerland, however, the production fluctuates between \(59.1 \cdot 10^6\) and \(68.8 \cdot 10^6\) MWh while the consumption increased from \(50.8 \cdot 10^6\) to \(57.5 \cdot 10^6\) MWh in the same time frame [4].

The electric grid is fed by several energy production plants, i.e. nuclear power plants, thermal power plants, renewable energy plants as well as storage power stations, i.e. water reservoirs. Although nuclear power plants provide a large amount of energy, they can not react on changing demands quickly. Renewable energy plants make use of given energy without emitting flue gas, but they rely on changing weather conditions and cause the need for compensation in the electricity grid keeping the load and frequency stable. Beside storage power stations, which are limited in capacity, the thermal power plants provide the ability to stabilize the grid independently.

The demand for electricity varies through the day and from the temperature of the season [134]. Before the liberalization of the energy market, the energy market underlaid an area monopoly. (Swissgrid exclusively manages the electricity grid in Switzerland since 2009.) Local provider sold energy to local customers. The liberalization of the market opened the market for trading energy production packages. In principle, the trading works as follows: The price and the amount of energy production for each plant...
6.1 Introduction and motivation

is based on the acceptance of a bid on the market for intervals of 30 minutes (National Grid Control Center, UK [134]). The liberalization increases the fluctuation of the energy price. In 05/2013 the Swiss-six base price of intra-day trading fluctuates between 15 and 50 Euro/MWh. This allows power plants, which can change their energy production quickly, to increase their income by fast adaption to changing market demands.

The effective power of the power grid in Switzerland is assured by a multilevel control. The primary control reacts within seconds by changing turbine set points and covers frequency deviations within 200 mHz. The secondary control is applied within seconds up to 15 minutes by plant running below their maximal capacity to increase and decrease energy production. After 15 minutes, the secondary control is relieved by the tertiary control. For tertiary control, which is applicable to waste-to-energy plants, the producer must adapt the energy output within 15 minutes. The overall load change ranges from -460 up to +510 MW [155]. Hence, even lowering the energy-production to stabilize the grid yields extra money.

The effective power control is traded as options on energy supply. A power plant sells the availability of injecting a certain amount of energy for a time frame into the grid by command without necessarily doing it. A bid contains a minimal amount of produced energy and a price for a time frame.

In general, the limiting factor for electricity output of heat managed waste-to-energy plants is the overall heating energy production and the district heating output. Due to the high caloric waste, the former is actually no limiting factor. Waste to energy plants already add sludge to the waste inflow of the furnace to reduce the heating value. The district heating output is, commonly, guaranteed by contract and depends on the weather. Thus, energy production of thermal plant does not exclusively depend on the weather like for a renewable energy plant. The thermal plant has the ability to decide about the energy excess on top of the district heating output.

To optimally stabilize the electricity grid, WtE plants technically provide the ability regulating tertiary control. Hence, it is necessary to know the capacities and the dynamics of the plant to optimally control it with respect to economic aspects.

6.1.3 Fuel market

WtE plants reduce waste to slag, flue gas and energy. In contrast to other fuel burning plants, such as oil, gas or coal, the waste combustion plant gets paid for taking fuel. WtE plants take domestic waste from the community that is mostly high caloric household garbage (in Switzerland), due to the high amount of plastics. A contract with the commune assures that the plant takes all accruing domestic waste at a fix price. Additionally, the plant takes industrial waste and sewage sludge on demand. The price is given by the plant operator. Due to competitors on the waste market, the gate fee of high caloric waste is lower than for less caloric waste, such as sludge.

6.1.4 Power plant structure

The plant is composed of four main components: The bunker, the combustion, the flue gas cleaning and the water-steam-cycle.

The bunker buffers waste from the delivery to the combustion feed. It decouples the waste inflow of the plant from the combustion feeding. The combustion transforms the waste under the addition of air into flue gas and slag. The flue gas cleaning removes hazardous substances from the flue gas before it is emitted into the environment. And the water-steam-process that transferrers the heating energy of the
combustion into the output energy of the plant, which is electric energy, district heating energy or process steam.

Figure 6.1 shown the energy flow from the waste market, through the incineration plant and into the energy market. The produced electric energy is reduced by the plant’s self consumption. It is the waste shredder, combustion air fans, pumps in the water-steam process (mainly the condenser pumps), and the flue gas cleaning (i.e. spray and electro static cleaning). The excess of electric energy is sold on the energy market.

The following section describes the working principle of the plant components bunker, combustion and water steam cycle. The flue gas cleaning is not relevant to the energy production of the plant. Due to environmental constraints, it continuously runs with full power. Hence, it is not part of the following consideration.

**Bunker**

The main function of the bunker is buffering waste. Industrial and domestic waste as well as sludge is delivered into the bunker. Bulky load might be shredded first. The outlet of the bunker is controlled by a crane. It feeds the combustion. The limitation of the bunker it its size. If the bunker is full, the plant can not take any further waste which causes financial losses (i.e. penalties). Although the bunker mixes the incoming waste, the crane operator can select the composition of the waste while feeding it into the combustion, which effects the caloric value and the heating output of the combustion. Due to the size of the combustion feeder, the reaction onto the heating value of the combustion is delayed.

**Combustion**

The combustion burns fuel. It converts air and waste into flue gas and slag. The released heating energy is used for steam production. It is transferred from the combustion chamber over a boiler into the water steam cycle. Hence, the flue gas is cooled down before treated in the flue gas cleaning. Combined power plants, additionally, utilize the heating energy of the combustion in a gas turbine and the water-steam cycle.

The combustion is controlled by the feed rate of air and waste. The waste is transported via the feeder onto a burning grid. The air is preheated and added from multiple directions by fans to the combustion process. Hence, mass flow and air temperature are controlled. The control is ecologically critical. Too little air causes non-optimal combustion which is indicated by carbon residuals in the slag and carbon-monoxide (among other part chemical reactions) in the flue gas. Too much air causes mono-nitrogen oxides ($NO_x$), due to the nitrogen and oxygen components in the air. Due to the required stability of
6.1 Introduction and motivation

the process, optimal combustion is run with a certain air excess rate. The controller measures flue gas (composition, temperature) and monitors the combustion with cameras to optimally control the process.

Although the produced energy is split up into multiple outputs, as it will be discussed in the next chapter, the main control of the amount of energy production depends on the combustion. However, stopping and igniting the combustion process is highly critical, although it is necessary due to revision.

Water steam cycle

The water steam cycle converts the heating energy ($Q_{\text{in}}'$) of the combustion into electric energy ($P_{\text{out}}$) and heating energy ($Q_{\text{out}}'$). The process cycle is shown in Figure 6.2.

First the feed water pressure is increased by the boiler feed pump. In the boiler the incoming heating energy heats up the water and transforms it from liquid to superheaded steam in multiple phases. The the water is preheated in the economizer and transformed from liquid to gas in the evaporator. The steam drum splits steam from liquid and assured that only steam is overheated in the superheater. After exiting the boiler, the steam pressure is relaxed and transformed into kinetic energy by a turbine and into electric energy by a generator. A condenser relaxes also the temperature of the steam until it is transferred into liquid. The energy is transferred into a cooling cycle and i.e. submitted into the environment (i.e. cooling tower or river). Finally, the deaerator assures that no gas particle are left in the water when it is fed to the boiler feed pump again.
Simulation is the representation of a real process as a mathematical model (equation) and its numerical or analytical solving to gain information about the process behavior [132].

The main advantage of simulation is replacing risky experiments although the quality of the results relies on sufficiently exact models and solving methods. Commonly, the purpose of simulating a power plant is the dimensioning of components and testing the behavior under different load conditions, such as chambers, boilers, heat exchangers and turbines, and the validation of the measurement units. In general, it allows investigating the system dynamics in order to find optimal parameters for controllers.

The models for simulation represent the simulated system in several languages, which differ, for example, in complexity or expressiveness. The model has a strong impact on solving or simulating the system.

Solving mathematical equations analytically is limited due to its complexity. Hence, numerical methods are used in computational fluid dynamics (ANSYS Fluent, ANSYS CFX [31]) or general solvers such as MATLAB [8]. Instead of explicit and exact solutions, numerical methods approximately solve models that are expressed as linear, non-linear, partially or ordinary differential equations (ordered by expressiveness, complexity and computational effort). Although solving linear equations cause less computational effort than non-linear equations, linear models do approximate the system behavior adequately only in a limited range, since natural processes are usually non-linear.

Power plant modeling and simulation is complex, since the underlying process is complex. 'A power station is a complex entity, embracing a wide range of what I refer to as primary disciplines - physics, chemical engineering, thermodynamics, mechanical engineering and electrical engineering' [134]. The models are systems of equations based on master and transport equations of mass, chemical change energy and impulse. Simulation is done by solving the equations.

The simulation of fuel burning power plants is focused on two main components: the combustion and the water-steam process.

Simulating the chemical combustion process is based on incineration equations which describe the reactions of fuel and air (oxygen and nitrogen) into flue gas and slag under emission of heating energy. Simulation is used to develop combustion controllers in order to reduce the risk of incomplete combustion, which results in toxic flue gas (i.e. \( NO_x \), \( SO_x \), \( CO \)). Nitric oxide, for example, is produced at high combustion temperature and carbon monoxide results from too little oxygen input. Computational fluid dynamics simulations focus on material flows to optimize the geometry of the furnace, such as the chamber. Simulating the combustion process can be done quasi-stationary, since the thermodynamic mass is negligible.

The second component, the water steam process, is composed of basis modules, i.e. heat exchanger, boiler, drums, with inputs and output mass flows that are changed by the component according to thermodynamic equations, such as heating energy exchange. Since some components have a thermodynamic mass, a adequate model considers the inter-stationary (or temporal) behavior of the system. The transient behavior of thermodynamic components, which defines the system dynamics, can be modeled as mathematical expressions [132]. A system component is modeled as physical time-space dependency between inputs and outputs as well as mass and energy storage.

Other components, such as turbines, can be modeled quasi-stationary [225] without significant loss of information.
6.1 Introduction and motivation

6.1.6 Optimal plant control

The operation of a power plant is controlled regarding two goals: The safe operation of the plant, due to the chemical processes in the plant, and meeting the demand on energy with respect to provision of the law and the efficiency of the production, due to the economical aspect of the plant. The former goal is critical due to the chemical processes in the plant that can not been stopped immediately.

The following section lists a selection of examples showing common control problems of power plants, which are controlled by simple but robust controllers such as PID controllers (see Section 6.2.3).

Waste combustion control

Generating power from combustion is primarily influenced and controlled by the produced thermal energy. In general, the combustion process transforms fuel (i.e. waste) and air into flue gas and slags. However, waste combustion is a complex and unstable chemical process due to the unsteady heating value from the changing fuel composition, which is hard to measure. The air and waste federate is controlled to maintain the flue gas temperature, bed height, and steam production based on sensor information about the oxygen rate in the flue gas, bed height estimation from under-bed air pressure and infrared thermography of the fire. Commonly, the multiple objectives of combustion control are to maximize steam production and waste treatment. However, the combustion is for example constraint by a lower limit of the flue gas temperature of 850 ° Celsius and an upper limit for the mass fraction of unburned carbon in the slags of 2% [45].

The combustion control defines the heating energy of the plant which is transferred through the boiler to the water-steam process. However, changing the set point of the combustion effects the overall heat production with a long delay.

Drum control

The drum is part of the boiler, which adds thermal energy to the cycle process. The purpose of the drum is to split steam and liquid water in order to provide dry steam to the turbine. To stabilize the heating process, the drum provides a feed water storage to balance between water supply and steam production. The objective of the drum control is to provide water to the boiler in order to match the evaporation rate.

If the level increases, the risk of getting (liquid) water into the superheater and the turbine increases, which will cause a damage of the components. If the level decreases, not enough water might be supplied to take the heating energy of the combustion, and the flue gas temperature increases which might damage the flue gas cleaning.

Hence, the goal of the drum control is to keep the water level of the drum at the midpoint of the vessel, which is not intuitively clear due to swell and shrinkage. Due to a pressure drop from increased steam demand, the drum level rises due to the increased volume of the mixture. The increasing drum level indicates to intuitively decrease the mass flow of the feed. This is misleading. The phenomena is called swell. Its opposite effect is called shrinkage. Hence, simply tracking the level of the drum is not sufficient to control the drum level.

Usually, a three-element feed water controller commands the feed value (inlet) and the steam value (outlet), as shown in Figure 6.3. The state of the drum is indicated by the level of water in the drum, the inflow (FI), the outflow (FO) and the pressure (PI) in the drum. During normal operation, a PID controller
takes care about the drum level. However, in extreme cases, the control is run manually by the operator, since predicting the system behavior is necessary for safe control.

Steam control

The heat energy that is transferred from the combustion to electric power mainly depends on the combustion process. The heat is used to produce steam which is relaxed by a turbine and transformed to electric energy by a generator. Since the effect of the combustion onto the boiler is highly delayed, the steam temperature at the boiler outlet is controlled by a spray-water attemperator. It feeds cool (liquid) water of the approximately same pressure into the overheated steam. The hot steam evaporates the cool water and the temperature of the mixture eventually decreases. As shown in Figure 6.4, the attemperator is located within the superheating section of the boiler. However, the control is limited by the saturation of the steam but immediately affects the overall steam production without delay.

Since the turbine is damaged by not getting dry steam, the amount of adding spray water is limited. However, adjusting the temperature of the superheated vapor, the overall energy output of the plant is controlled within the physical limitations of the turbine.

Heating control

The superheated steam is relaxed in two stages. The pressure is relaxed by a turbine and the temperature is relaxed by a heat exchanger. The turbine transforms the heating energy into kinetic energy which is used to impel a generator that produces electric energy. The heat exchanger condenses the water transferring the heat into a separate cooling cycle, which emits the energy to the environment (cooling tower or river) or public heating.

In principle, the turbine and the heat exchanger are organized sequentially. Real plants, however, provide multiple turbine stages and intermediate extraction, as shown in Figure 6.5.

Bypassing and intermediate extraction allows load balancing between heating and electric energy output.
Although most of the energy is extracted in the first stage of the turbine, it enables to direct the mass flow into the district heating heat exchanger instead through the second turbine stage. Moreover, the turbine bypass allows to completely pass the turbine and relay the steam at the condenser, which is used in the starting phase of the plant. In any case, the steam is eventually condensed to liquid by the condenser before it reaches the deaerator and the boiler feed pump (see Figure 6.2).

A difference between turbine and heat exchanger is the reaction time. The electric energy production can be changed within minutes, since the turbine reacts relatively fast onto load changes. The reaction of the heat exchanger of the district heating, however, is slow, depending on the thermal mass of the component.

The energy output of the power plant is controlled in two ways: The overall energy production and the energy split. Changing the overall amount of energy production is controlled by the combustion (Section 6.1.4) and the steam control (Section 6.1.6). The split of the energy into electric power and heating energy is done at the water steam cycle by the bypass valve and the split value.

The district heating production is relatively slow and predictable. Today, the weather forecast allows predicting the needed overall amount on heating energy precisely, although the public heating network might be fed by multiple plants with changing contributions and changing the mix would effect the contribution of single plants although the overall production is not changed.

As discussed in Section 6.1.2, the energy grid is, compared to the heating network, more dynamic and hard to predict. Being able to rapidly change the energy production is an advantage on the energy market. And controlling the plant according to the market demand with respect to its physical limitation and safe operation is an ongoing research topic.

The safe operation of the plant with minimal pollution and optimal efficiency are the important goals of controlling a plant. Since waste incineration plants get money (gate fee) for their fuel, the focus of such plant was to optimally burn the waste. Producing energy was useful to supply the plant components, i.e.
6.1.7 Environmental analysis

This section summarizes the results of a *SWOT analysis* [36] for a WtE plant. As a method for environmental analysis of a project or institution, a SWOT analysis investigates the strengths, weaknesses, opportunities and threats of an institution. Strengths and weaknesses are driven and controllable by the institution, whereas opportunities and threats are exogenous and uncontrollable. The goal of the institution is to reduce the dependencies on threats, get use of opportunities, reduce weaknesses and increase strength.

In general, WtE plants developed from waste-reduction to power-production plants. This extends the interfaced markets from only the waste market to the electric energy and district heating market.

The *strengths* of a WtE plant are that the plant can buffer energy. The fuel (waste) is stored in a bunker. Hence, the plant can change the load of power production by changing the heating value of the waste inflow of the combustion. This is an advantage compared to i.e. nuclear power plants. The second advantage of burning waste is independence of the weather. Compared to wind farms or solar power plants, the energy production of WtE plants does not depend on the weather. In contrast to other fuel burning plants, WtE plants get paid for taking fuel. Low heating waste has a high gate fee and low heating a high gate fee.

The *weaknesses* of WtE plants are the low amount of energy production. Compared to a nuclear power plant, the waste incineration plant does not influence the energy market. Moreover, the energy level (heating value) of the fuel is highly uncertain, which makes the combustion process unstable as well.
as hard to control and to predict. Starting and stopping the combustion process is, for example, done manually and only for revision work due to its complexity.

The opportunities of a WtE plant are the increasing heating value of the waste. Due to the high amount and the high heating value of plastics, common waste is, today, highly combustible. The second opportunity is the liberalization of the market for electric energy. The WtE plant can benefit from getting paid for rapidly changing the production of electric energy.

The threads of a WtE plant are competitors on the waste market, which influences the heating value of the fuel and the gate-fee incomes. By contracting with local district heating providers, the WtE plant assures to get paid for energy production but it also gets dependent on the weather. The energy transfer to the district heating network is demand driven and the amount on needed energy mostly depends on the weather. Another thread is the high heating value of waste. The furnace is designed only for a certain window of waste heating value. Leaving this window makes it impossible to process the given waste due to physical constraints. The last thread is the changing price for electric energy production. Not knowing the income for electric energy production makes it hard to calculate the price for taking waste in order to defray (operating) costs and assure profitability of the power plant.

**Approaches**

The following section summarizes approaches to increase strengths, decrease weaknesses, reduce the dependencies on threads and get use of opportunities.

The strength of buffering fuel can be utilized as follows. In general, the gate fee of high energy fuel is lower, since it leads to a higher energy production and income. Low energy fuel, on the other hand, gets more gate fee. The electric energy market, however, might not only pay for produced energy but also for reducing the energy production due to controlling the network stability. Hence, taking low energy waste, which decreases the overall energy production, might eventually be more valuable than high energy waste. Due to the bunker, a WtE can buffer waste of different qualities. Due to the independence of the weather, a WtE can compensate power grid load changes caused by renewable energy plants. It might get paid not even for actually applying but just being prepared to stabilize the electric grid.

The weaknesses of the low amount of energy production can be reduced by increasing the number of production lines. The unstable heating value of the waste heating value can be handled by extending the measurement systems, i.e. small test burning unit, and robust control methods (see Chapter 6.6). Starting and stopping the combustion process is, up to now, still a challenging task.

The opportunity of increased heating value is addressed by additionally accepting waste that is almost not combustible and has, therefore, high gate fee. Mixing highly combustible waste with bad burnable sludge results in a appropriate fuel mix for the incinerator. Additionally, the mixture’s heating value is controllable. The liberalization of the energy market results in changing prices and new business models, such as selling the option to in- or decrease the energy production in order to stabilize the grid. However, to benefit from the new market situation, the KVA operator needs to be well trained in order to know the plant’s behavior in detail.

Due to the need to fulfill the contracts with the district heating provider, the WtE plant depends on the weather. The weather forecast can be used as indicator for the needed amount of energy in the heating network, although the mixture of multiple producers is still unclear. The needed amount of heating energy is subtracted from the overall production to calculate the electric energy. Like short-term prediction of the power production from wind farms, it relies on the weather forecast. The correlation between wind and energy production is reasonable. However, the system dynamics of the cross effects
between electric and heating energy production load changes of a combustion plant are more complex and crucial for giving promises to the electricity grid provider. Turbines react fast and heat exchanges slowly. Investigating the optimal split factor between district heating and electric energy production is an open question.

Modeling the energy market allows predicting the demand on electric energy. This is common on monthly down to daily bases. The resulting price, however, depends on the market player and is hard to predict. The thread of increasing heating values is discussed above. By controlling the inflow mixture of the incineration, the heating value and the combustion is controlled. However, defining the optimal gate fee price is an open question.

### 6.1.8 Sequential decision making for power plant control

As discussed in Section 1.2, sequential decision making under uncertainty provides a method for optimally controlling a process with respect to uncertainty. To justify the application of such a complex control method, the given problem must show three aspects. It must be a multi-objective optimization problem and the controlled system must be dynamic and uncertain.

Optimizing the energy production of a power plant is a multiple objective problem, since multiple demands are served. The plant produces electric energy and heating energy.

A power plant is a dynamic system, due to the thermodynamic effects of the components, such as turbines and heat exchangers. For example, a turbine reacts faster on energy changes in the water-steam-cycle than heat exchangers, due to their thermal mass. Controlling the process with valves is physically limited. Hence, considering the temporal behavior in terms of applying strategic control is valuable in controlling the system optimally.

Uncertainty comes from the plant’s sensor and actuator errors and the instable heating energy of the combustion, due to the inhomogeneous waste composition. Hence, applying strategic control with respect to uncertainty is reasonable to optimally control the plant.

To deduce a non-myopic control policy, however, model predictive control relies on a system model. The system power plant, however, is highly complex. Hence, modeling the power plant is a challenging task.

### 6.2 State of the art

This chapter introduces the state-of-the-art of waste to energy plant simulation and control techniques. The common methods of operational plant control are based on simple and robust methods, such as PID controller. Assistant systems, however, are based on advanced control methods, i.e. MDPs. They do not autonomously control the plant but provide suggestions to the operator. So far, such systems are used in extreme situations and for operator training.

#### 6.2.1 Waste to energy plant simulation

The simulation of waste to energy plant or - in general - power plants considers two main components of the plant: the combustion and the water-steam process. Although the flue gas clean is highly important when burning waste, the following section focuses on the core components, due to economical considerations.
Simulating the combustion process is based on incineration equations which describe the chemical reactions of fuel and air (oxygen and nitrogen) into flue gas and slag under emission of heating energy. Bardi presents a model and optimal control of a waste energy plant focusing on the combustion process [44].

The water steam process is composed out of basic modules, i.e., heat exchanger, boiler, drums, with inputs and output. Zindler presents an object-oriented approach for simulating dynamic power plants [225] focusing on the thermodynamic components on the water steam cycle. Each component (object) has inputs and outputs in terms of mass flow, temperature and pressure and energy. The components are parametrized, i.e., the heat transfer of a heat exchanger. Basic equations, on the basis of conservation of energy and mass, express the (quasi-stationary) thermodynamic behavior of the component and more complex differential equations describe the delayed system response on input changes even more accurately.

**Thermal power plant simulation tools**

Since simulation is mainly focused on the dimensioning of single components, manufacturers develop specific simulation software which cannot be easily integrated with other simulation components.

Simulation toolboxes allow setting up a plant model by integrating the single components. Thermal power plant toolboxes for MATLAB/Simulink® are provided by S.A.T.E. Systems and Advanced Technologies Engineering S.r.l. and EUtech Scientific Engineering. S.A.T.E. provides power systems and separation process simulations for oil processing, coal and gas plants, based on a MATLAB/Simulink® architecture. EUtech provides Thermolib, a toolbox for MATLAB/Simulink® for modeling and simulation of thermodynamic systems. ‘It is based on fundamental thermodynamic properties and state calculations and provides numerous components for setting up an entire thermodynamic system including components like pipes, heat exchangers, pumps and compressors, chemical reactors, burners, tanks, valves, splitters, mixers and more complex subsystems like fuel cell stacks etc.’ [24]. Thermolib provides chemical reactors to simulate combustion processes as well as a set of thermodynamic components, such as heat exchangers, pumps, compressors, turbines, tanks and valves, to model the water steam process. It provides multiple media flows, i.e., CO₂, H₂O and several refrigerants [24]. One major advantage of the software is the signal-bus (flow-bus) to connect model components. It enables transmitting the properties of a media flow (pressure, temperature, mass flow, enthalpy and vapor fraction) between the components.

Based on such simulations of the power plant, controllers can be tested and optimally parametrized without causing any damage on real systems.

**6.2.2 Intelligent control in the energy sector**

Metaxiotis et al. summarize the application of intelligent computer applications in the energy sector from 1990 to 2003 [145], i.e., energy planning and load forecasting [146], alarm processing and system diagnosis. They compare expert systems, neuronal networks, generic algorithms and fuzzy logic. Vale et al. present and intelligent alarm processor for power system control centers [211, 210]. They propose a knowledge-based operator assistant for decision support during incident conditions, due to the large amount of incoming messages which can not be handled manually.

Jebaraj et al. review energy models in terms of energy supply, demand and supply–demand models as well as forecasting models to, for example, predict load peaks or weather depended energy demand [113].
In contrast to controlling and assisting energy distribution, the operation of a power plant focuses on the physical aspects of generating energy. The following chapter describes the control aspects of running a power plant.

6.2.3 Power plant control

To safely run the power plant, controlling the facility focuses on the physical aspects of components. As described above, controlling the feed water of the drum is considered separately from setting the outlet temperature of the boiler.

For industrial control, fast, reliable and robust controller systems are required. Hence, a widely used approach in plant control are proportional-integral-derivative (PID) controllers. Such model-free controllers work well on continuous systems to reach a given set point. As shown in Figure 6.6, the controller acts on the system (plant) and gathers a feedback of the system state by a sensor. Comparing it to a given set point gives the deviation (error) between the plants actual value of the controlled variable and the set point. The controller internally manages the proportional, the integral and derivative values of the error and selects the control value according to a weighted sum (static parameters of the controller) of the error values.

As an example, the temperature of the boiler’s steam outlet (see Section 6.1.6) is managed adding liquid water to the hot steam. The control variable is the value position which controls the mass flow of the liquid water. The control variable is set according to the measured temperature. If the measurement is below the set point, the valve is closed and vice versa.

However, one drawback of such controllers is the unintuitive parameter tuning, which is mostly done automatically since manual tuning methods can be relatively inefficient. Moreover, the model-free controller does not take into account the limits of the actor (controlled variable). For example, an incremental controller commands a change in the position of the controlled device. If the controller and actuator, however, are considered together as a system, it will be seen that the error will still occur when the actuator reaches the limit of its movement. Expressing non-linear system correlations as linear matrices in the PID controller is, moreover, only valid for a limited window around the optimal operation point. Hence, this family of controllers can not handle large adjustments, such as sudden load changes. Moreover, without signal filtering such systems are prone to measurement errors, since the controller immediately reacts to the measured value.

Model-free controllers, moreover, do not take into account long term (strategic) effects of the control onto the system. They might not consider the system dynamics, i.e. overshooting, as MPC controllers (see Section 2.3) do.

In general, PID controllers are widely used in industrial control for keeping system variables around a certain set point. Such controllers are simple to implement, although unintuitive to parametrize. Uncertain measures are stabilized by cascading sensors or filtering the sensor signal. To deduce control laws for operating commercially optimal, the control system should additionally take into account the dynamic market demand. Hence, applying model predictive control is essential to optimally operate a power plant regarding economic aspects.

6.2.4 Waste incineration plant control

Bardi et al. [45] claim that the combustion process is, today, manually driven by the operator in most waste to energy plants. To control the plant automatically, Anderson et al. propose a multi-objective
strategy in order to optimally run a waste incineration with respect to maximize throughput (reward for taking waste and producing energy), the loading on the gas-clean-up system (penalty costs due to pollution) and other operational constraints. However, due to the complexity and uncertainty of the combustion process, advanced, reliable and robust controllers are needed. Chen et al. [64] present a generic, algorithm and neuronal network based controller to manage the many uncertain factors and reduce the operational risks. Another solution based on advanced fuzzy logic technology is demonstrated and implemented by Krause et al. [123].

Bardi and Astolfi present a detailed process model and optimal control for waste incineration [45]. They propose an approach for automatically controlling the waste bed temperature with robustness against changing waste composition, that is based on input/output linearization and extreme point seeking. They investigated optimal reference temperature (set points) for maximizing steam production. However, due to changing energy demands, optimal steam production is not necessarily the maximum steam production. Although a load change can be handled in the water steam process, the overall energy production of the plant is driven by the combustion. Changing the control variables of the combustion, however, effects the steam production with delay. Hence, demand driven energy production must consider the water steam and the incineration process.

As an add-on to the existing Advanced Combustion Control system, ABB provides with WACS Grate Combustion Optimization [1] a module to optimize the waste incineration on moving grates based on MPC techniques with respect to multiple goals. Another module is the WACS District Heat Forecasting[1]. It aims to predict the demand of district heating energy in order to optimize the overall energy production and excess for electric energy disposal.

Due to the uncertain nature of the combustion, probabilistic methods are naturally selected for modeling and optimizing such systems. Due to the delay of combustion and thermodynamic elements, model predictive control techniques are used. However, to reduce the risk of automated control, assistant systems suggest a control action to the operator, who eventually executes the action.

### 6.2.5 Power plant assistant systems

Controlling a power plant is a complex process. Extreme situations of power plant operations are unusual and abnormal situations. They are critical in two aspects. First, automatic control can hardly deal with extreme situations, since the controller are configured around a certain set point which is not given by definition of an unusual situation. Hence, the plant is commonly run (partly) manually. Moreover, extreme or abnormal situations rarely occur. Hence, the plant operator’s experience does not cover such dangerous situations and the operator might react wrong in untrained situations, due to stress. Under extreme conditions, an intelligent assistant system becomes a helpful tool in order to normalize the operation.

Intelligent assistant systems do not autonomously control the plant but give rational recommendations.
to the operator which finally decides to apply the action. Elizalde et al. demonstrate the potential of an intelligent assistant for operator training [78]. It gives recommendations as a sequence of actions for optimal steam generation which are deduced from automated planning. The operators final action is monitored and captured as user behavior model, which can be used as uncertainty in applying the proposed action.

AsistO, an intelligent assistant software for power plant operators, is presented by Reyes et al. [178]. It provides recommendations (a sequence of actions) based on decision theoretic planning and probabilistic reasoning, in particular factored MDPs and POMDPs.

The software components are a recommendation system, a diagnosis system and an explanation system [176]. It is run in either operation assistance mode and gives recommendations to the operator, or in training mode where it learns the behavior of the controlled system. The software components are the user interface, plant simulation, data management, model management, modeling and planning.

The explanation system provides simple, predefined explanations as reason for actions. It is found in a user study that assistance improves the learning progress [176].

The operation assistance is applied to optimally control the steam generation of a power plant (see Section 6.1.6). The software controls the water level in the drum of the heat recovery steam generator by changing the set point of PID controllers for the feed water valve (inlet) and the main steam valve (outlet). The MDP controlled is applied to appropriately react to heavy load changes of the electric energy demand [176].

The sequential decision making process is modeled as MDP based on a curve of recommended operation of the boiler regarding drum pressure and mass flow. The MDP selects the optimal control out of 5 actions to open and close the valves. The factorized state (space of 384 states) is composed out of the feed water mass flow (Ffw), the main outlet mass flow (Fms), the drum pressure (Pd), the power generation (g) and a disturbance (d) [175]. The state space is discretized manually and the transition matrix is learned using the K2 algorithms of Elvira [69], a Java tool to construct probabilistic decision support systems, from 260 hours of operation [33]. The tool models a Bayesian network by influence diagrams (nodes, links and relations). The reward model is obtained from expert knowledge considering the state aspects $D_p$, $F_{ms}$ and $g$ [33]. Reyes and Spaan propose to use POMDPs [42] extending the model by observations and an observation model, due to the fact of uncertain sensor values. The observation matrix, which is also obtained with expert knowledge, is given by Agueda et al. [34]. The control policy is deduced using a proprietary solver which converges after 10 iterations (discount factor 0.9) within 1 second [179] and for a discount factor of 0.3 also in 1 second [177].

The recommendations are optimal control policies which can be deduced from plant models using automated planning (see Section 2.3).

Deducing policies, however, relies on modeling the controlled system. Since power plant are complex and the system behavior must be expressed accurately, power plant model are also complex. As discussed in Section 4.2, for non-trivial modeling of MDP and POMDPs the action, state and observation space grows rapidly and the model becomes difficult to manage.

6.2.6 Research gap

Operating a power plant is complex due to the nature of the plant. Today, simple but robust controllers are mostly applied to independently operated components of the plant (see Section 6.2.3). Such controllers, however, are set up to control the system around a certain set point. If the system leaves the operation
6.2 State of the art

window, manual control by the plant operator is necessary. To control the complex system, the operator needs knowledge in many areas such as chemical, physical, electrical and mechanical engineering. Hence, assistant systems are developed to make suggestions and to train the operator.

It is shown by Agueda et. al. [33, 34] how to apply POMDP to power plant control. The controller suggests an action to the operator in order to optimally control the water level of the steam drum of the boiler in the water-steam process. The finding are integrated into the recommendation module of a simulation-based training system [176, 178].

Agueda et al. [34] motivate the application of POMDP, by showing that the given problem is a multi-objective problem under uncertainty. They motivate the measurement uncertainty by sensor noise and decalibration, and the uncertainty of the system dynamics by unknown load demands. Beside the demand, which is model as disturbance, the state space covers physical aspects, such as mass flows and power generation of the plant. However, uncertainty is only reasonable for the disturbance aspect of the system dynamics, which is not controllable.

The underlying model is presented in detail by Agueda et al. [34, 33]. The state and action space are defined manually. The size of the action space is minimized by defining step-wise changes instead of absolute values. It is claimed that the transition model is obtained from measurement data during minimal, maximal and average load of a thermodynamic power plant running 736 hours. The observation model, which is an essential part of the motivation for applying POMDP to this problem, is based on human experience or could be learned from historical failures. It is claimed by Reyes [178] that the model can be deduced using existing learning software, such as Elvira [69]. The reward model is also based on human experts. It covers the riskiness of applying actions in certain situations (states). Generally, the model covers the physical aspects of a specific part of the water steam process as one component of the power plant.

Manually modeling the system as POMDP is not intuitive and error-prone. Hence, allowing process experts to model the given problem is a high-level language would increase the quality of the model and the resulting control policy.

It is, moreover, hard to motivate the reward function, which is essential for the optimization, based on physical system characteristics, like attrition or damage of technical components. Covering economical aspects, like profit from selling energy, is reasonable in order to run the plant optimally. However, including economical aspects implicates modeling not only a single component, like the boiler, but the overall power plant, such as a waste-to-energy plant.

Today, the waste is high caloric and the energy market is liberalized. Hence, it is possible to make money by producing energy at the right time and the right amount. Even not producing energy yields money, due to the need of stabilizing the electric grid. Controlling a power plant got a new dimension: the commercial aspect.

Demand driven power production depends on the market and causes the need for an additional competence: Being an expert on selling energy. WtE plants have the ability to serve tertiary control and stabilize the electricity grid. To make extra money, tertiary control requires to in- or decrease the electric energy production within 15 minutes.

However, after heavily changing the operation point, stable operating can not be guaranteed by simple, i.e. PID controllers, which are set up for a specific quiescent point. Due to the nature of waste as a fuel, the combustion process is difficult to predict. Compared to a gas plant, the energy production is not stable. This makes it difficult to control a waste to energy plant in a commercial way. To strategically control the system in an optimal and save way, the system’s dynamics and uncertainty must be taken into account.
Model predictive control, such as MDPs and POMDPs, allows setting up strategic control policies of dynamic systems with respect to uncertainty. They adapt the system step-wise with respect to a multi-objective cost function (see Chapter 2). And, as introduced in Section 2.6, Reinforcement learning allows dealing with changing environments under uncertainty, which is given by the inhomogeneous waste composition and heating value.

However, model based controllers are complex and rely on realistic models of the specific system. To correctly express the behavior of a highly complex system, such as a thermal power plant, the physical aspects must be taken into account. Due to the complexity of the power plant, the dynamics of the plant are hard to model manually. Setting up a probabilistic Markov model is even more difficult, which causes errors in the model and in the resulting policy. But controlling a power plant is critical and errors must be minimized.

In general, it is dangerous to explore the behavior of a real system, due to the nature of the power plant. It is expensive to damage the plant or underperform and not fulfill the supply agreement for energy. Modeling the plant’s components separately, instead, simplifies the modeling process and reduces the risk of errors. Simulating the plant based on a model allows exploring the behavior of the plant to figure out new capacities, which might give the plant manager an advantage at the market.

Due to the dynamic behavior of the plant, an optimal control policy requires deliberation, which is hard to deduce manually due to the complexity of the plant. Automated planning, an area of Artificial Intelligence, studies the computational planning process. To develop non-myopic control policies for demand driven energy production, in particular tertiary control, automated planning methods are applied in the following section.

6.3 Contribution

The liberalization of the energy market causes fluctuating energy prices due to changing environmental conditions for producers and demand. Fast adaption to changing prices and demands enables to increase the income for energy producers such as WtE plants. Due to the reaction time of 15 Minutes, WtE plants can contribute to the tertiary control of the Swiss electricity grid. The tertiary control is based on a bidding process. A energy producer offers a bid defining the electrical power in MW, its price in CHF/MW and the time frame.

In contrast to common incineration plants, WtE plants get payed for fuel, which is in particular waste, higher income from energy production enables lower gate fees and results in competitive prices for industrial waste as well as reduced waste taxes for the domestic waste.

In contrast to regenerative energy produces, waste burning plants do not depend directly on the weather situation. Additionally supplying heating energy to the local district heating network (District heating supply is commonly contracted), however, makes the plant depended on the weather, since the district heating demand depends on the temperature. Nevertheless, the load change of electric energy production reacts faster than the load change of heating production. Hence, it might be possible to quickly change the electricity production without effects on the district heating supply.

The upcoming question is: How to control the power plant economically optimal? In particular, how to control the power plant in order to quickly adapt on changing market (i.e. fulfill tertiary control and district heating supply) with respect to uncertain system dynamics? What are the physical reasons that justify uncertain system dynamics?
This chapter describes a system that automatically deduces strategic control actions for changing market situations. Based on the reward model, which models the fee and demand of the markets, the software calculates the optimal policy and answers if and how to reach the target point. To adapt the control policy to changing market situations, Reinforcement learning methods are applied to deduce the control policy. The learner continuously adapts the control policy to model changes, in particular the market price and demand.

The chapter is organized as follows: First, the waste to energy plant model is developed and parametrized with data of a real plant. With regards to controlling the system, the model components are discussed in detail in terms of input and output interface and control variables.

Then, the problem characteristics in terms of uncertain system dynamics are discussed in order to motivate the application of Markov Decision Processes and Reinforcement learning. In order to connect the controller with the simulation, the input/output interface between artificial intelligence and plant simulation is defined.

Learning controllers deduce optimal policies and are, eventually, compared with regards to economical aspects of the plant control before the chapter concludes with a summary.

### 6.3.1 Waste incineration plant 'KVA Turgi'

This section describes the waste incineration plant facility KVA Turgi located in Turgi, canton Aargau, Switzerland. As shown in Figure 6.7, the power plant is structured in the components *waste reception*, *bunker*, *combustion*, *flue gas cleaning* and, finally, the *water-steam cycle*. Each component is described in detail in the following section.
6 Case Study: Strategic power plant production control

Waste reception

The waste reception is located at the beginning of the process chain. At this stage, waste from the regional households (domestic waste) and industrial waste enters the plant via municipal garbage collection service or private transport. Before it is filled into the bunker, which is described in the next section, the truck load is weighted to determine the gate fee (reward).

Bunker

The Bunker stores waste. It is filled up with waste from the reception and feeds the combustion by a crane. Waste is buffered on several stacks from hours to days until it is fed into the combustion. Hence, the bunker decouples the waste inflow of the plant and the combustion.

Combustion

The combustion burns waste on a constantly moving grate. Waste and air is fed into the furnace and flue gas as well as slag exit the combustion component. The flame temperature is around 1400 K. The flue gas exits the chamber through a heat exchanger, which cools it down by transferring the heating energy into the water steam cycle, before it is eventually cleaned in the flue gas cleaning.

Flue gas cleaning

Due to strict emission regulations, plant managers are forced to sophisticatedly clean the flue gas produced by the combustion process. The exhaust gas mainly contains carbon, nitrogen and sulfur oxides and gaseous water. To reduce the amount of monoxides to an environmental harmless level, a complex after treatment of washing and filtering processes is applied. Due to the complexity of the process, flue gas cleaning will not be discussed in this thesis.

Water steam cycle

The heating energy of the combustion process, which is carried by the flue gas, is transformed into electricity and district heating by the water steam process. Liquid water under high pressure of approximately 50 bar is heated up in the flue gas boiler and evaporates. Then, its pressure is relaxed by a two-stage steam-turbine; both stages are connected to a generator, which produces electric energy. After the first stage, the steam can either be led to the second stage or a heat exchanger, which feeds the local district heating network. In any case, the remaining vapor is condensed back to liquid water by cooling water from the local river and pumped back to the flue gas boiler.

6.3.2 Model

As discussed in Section 6.3.4, strategic control policies can be deduced from existing models in two ways. Either a POMDP model is deduced from sampling the model, which is then be used to deduced a strategic control policy by planning, or a control policy is directly developed through Reinforcement learning. Compared to learning, the first approach has two disadvantages. First, the effort of separately running
model deduction and planning, and second drawback of not using the already gathered knowledge in order to focus the model deduction process from the environment.

In the context of WtE plant control, this environment is the plant and its associated reward mechanism. As discussed in Section 6.1.5, for practical and safety reasons performing research by applying learning controllers to a real WtE plant is infeasible. Thus, a simulation of a power plant is build in order to develop a control policy by a learning agent.

In the field of control engineering, MATLAB finds wide use for modeling and simulating dynamic systems. To simplify modeling, MATLAB/Simulink® extends MATLAB by a graphical user interface: "MATLAB/Simulink®, is a block diagram environment for multi-domain simulation and Model-Based Design. It supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems. MATLAB/Simulink® provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB, enabling you to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis." [143].

The plant is composed of components (MATLAB/Simulink® blocks). Each component defines an internal structure and parameters as well as the interface that interconnects the components and the controller. The input and output interface of the modules defines the interconnection in terms of energy- and mass-flow between the components. Moreover, the controller connects with the component’s decision variables. In order to set up the model realistically, the component internals (structure and parameter) are based on measurement data from the real power plant KVA Turgi, i.e. the thermal mass of components of the water steam cycle.

The model is composed of the plant and the market. The market components model the waste procurement market and the energy selling market in order to set up a reward model. The plant is divided into 3 components: The bunker, the combustion and the water-steam cycle, according to the general structure of power plants (described in Section 6.1.4).

Conceptually, the energy- and mass-flow is modeled as follows (see Figure 6.15): The waste market feeds waste to the bunker, which is defined as mass flow (scalar) and composition of elements (vector). The bunker feeds the combustion, where the waste is burned. The combustion heat is transferred into the water-steam cycle, which drives a turbine to produce electric energy to the electricity grid and heat exchangers to provide heating energy to the local network. Depending on the energy demand and the amount of waste as well as the penalties and rewards the cost function is modeled as reward model.

In the following sections, the components are described in terms of structure, parameter, input/output interface and decision variables. The I/O interface defines the state- and action mapping between simulation and artificial intelligence controller.

**Bunker**

In concept, the bunker represents a buffer for physical mass. It is fed with incoming waste from the waste market and emptied by the crane that feeds the combustion input.

The parameter of the bunker is the size, which is used to simulate a bunker overflow and cause the plant to reject incoming waste from the market which effects the cost function, in terms of penalties for rejecting domestic waste and not getting gate fee for industrial waste.

For simplicity reasons, the bunker is represented as single pile with optimal waste mixture. Technically, the bunker smooths the waste composition and heating value on multiple, different waste inflows. In
principle, a bunker pile is implemented as an integrator that accumulates the incoming waste flow subtracted by the outgoing waste flow, as shown in Figure 6.8. If the bunker is empty, the waste outlet is set to zero. If the bunker size is reached, the inflow is ignored and an indicator at the output interface is activated.

The bunker outlet feeds the combustion inlet, which is described in the next section.

**Combustion**

The combustion module represents the chemical transformation of air and solid waste into slag and hot flue gas, according to thermodynamic processes. Due to the thermolib modeling toolbox, the outlet of the combustion is a single mass flow. Its heating energy carries the overall energy production of the plant. It is controlled by the mass flow and the fraction of the two inflows waste and air. The waste composition is defined according to the bunker outlet and the overall mass flow is a free decision variable, which is limited by the bunker fill. As shown in Figure 6.9, the second inflow - air - is modeled as static composition of nitrogen and oxygen. The air temperature is set to 293 K and the pressure is set to 1 bar (according to the environmental pressure). The mass flow is automatically set according to the carbon fraction of the waste with a stoichiometric rich mixture of $\lambda = 1.06$. Hence, the free control parameter of the combustion is the inlet mass flow of the waste.

The heating energy of the outlet steam is transferred to the boiler of the water steam cycle, which is described in the next section.
Water steam cycle

The water steam cycle transforms the combustion heating energy into electric energy and district heating energy, as introduced in Section 6.1.4. In the closed cycle, the pressure of the liquid water is increased by a pump before it is heated in the boiler (see Figure C.2). Then, the superheated steam is firstly relaxed in terms of pressure by a turbine, which drives a generator that produces electric energy, and then relaxed in terms of temperature by heat exchangers, which feed the district heating network. Eventually the condensation of the water is assured by a heat exchanger that is connected to the environment (cooling tower or river).

The structure of the water steam cycle represents the plant KVA Turgi and the components of the water steam cycle are parameterized accordingly, i.e. the thermal mass of heat exchangers. The pump pressure is set to 50 bar. The water input of the district heating network is statically set to 12.5 kg/s and a temperature of 293 K at ambient pressure. The condenser input stream is set to a mass flow of 695 kg/s with a temperature of 292 K at ambient pressure. The outlet pressure of the 1st turbine stage is set to 3.5 bar and the outlet of the 2nd turbine stage is set to 1 bar. The isentropic efficiency of the turbine is set to 0.7. The electric power production is deduced from the kinetic energy of the turbine by a generator. The overall energy production is the power group production subtracted by the power consumption of the pump. The district heating power production is deduced as offset between the inflow and outflow energy of the district heating heat exchanger.

The water steam cycle control variables affect the overall energy production and the energy split into electric energy and heating energy. The overall energy production is controlled with the steam outlet of the boiler. As discussed in Section 6.1.6, the steam temperature is controlled with spray water injection (see Figure 6.4 and C.2). The energy split, however, into electric and heating energy is controlled by two variables, as illustrated in Figure 6.10. In practice, the turbine bypass enables bypassing both turbine stages and directly driving the condenser. This setup is mainly used when the plant is started up or brought down. If the combustion does not provide enough heat to superheat the water steam, the turbine might be damaged from wet steam. However, bypassing the turbine and the district heating heat
Figure 6.11: Waste component (MATLAB/Simulink®)

exchanger allows to not output energy to the markets but only to the environment. The second control variable is the split value, which is located between the two turbine stages. It controls the steam transport to either the second turbine stage, in order to produce electric energy, or to the heat exchanger, in order to supply district heating. Implementation details are listed in Appendix C.

The input of the water steam module is the heating energy of the combustion. According to the control variables, this energy is transferred into the output parameters electric energy and district heating energy. The process is controlled with respect to a target electric power and district heating production, which is indicated by the heating energy added to the district heating network, see Figure 6.10.

The water steam process concludes the description of the WtE power plant components. Next, the markets of the plant are described in detail. The main difference between plant and market is the controllability. Whereas the physical plant behavior can be controlled directly by i.e. valves, the market cannot be influenced in general.

The market models mainly cover the economical aspects of the power plant. The overall reward of the plant is add up by reward models for each production output, i.e. electric energy, district heating power production and gate fee as well as waste rejection penalties.

Waste Market

The waste market represents the disposability and the price for different waste compositions. Waste with a high heating value gets less gate free than waste with a lower value. Three different waste compositions are modeled chemically: domestic waste, industrial waste and sewage sludge.

The implementation of the waste composition block in Simulink is shown in Figure 6.11.

The economical aspects of the waste market are modeled as follows. The gate fee model is a simple multiplication of mass and price of the taken waste per time step. And the waste rejection penalties are also a static costs times the rejected mass, which is activated if the bunker is full.

The procurement market the waste occurrence and gate fee for sludge and industrial waste as well as
penalties for refusing delivery of domestic waste. This is modeled as part of the reward model.

**Energy Market**

The sales market defines energy demand and prices as well as penalties for not fulfilling the demand. The energy markets of a WtE plant are the electricity market and the district heating. The district heating price is, in case of the KVA Trugi, bound to the price for electrical energy. As already stated in Section 6.1.2, the demand for district heating depends on the weather.

The demand and price for electric energy production, however, differs on hourly basis. Meeting a given demand is the goal of the plant controller.

The demand of the district heating and the electric energy market is modeled as getting rewards, if the demand is met by the plant, and paying penalties otherwise. This is motivated by the fact that the plant might buy energy from other plants in order to fulfill the contract for energy production.

Figure 6.12(b) shows the implementation of the sales pricing model in MATLAB/Simulink®. The inputs are the energy production and the demand; the output is the reward (profit rate). The parameters of the model are the interval width and the reward and penalty value. The sales pricing model represents getting reward if a certain energy production demand is met. The demand is defined as interval, since the amount of energy production cannot be met exactly. If the power plant produces the requested amount of energy, it earns a reward per time step. If not, it pays penalties (negative reward), which are higher that the reward for production, since the plant must buy it from another producer in order to fulfill the contract.

**Parametrization**

The plant model is parametrized according to the real plant KVA Turgi, introduced in Section 6.3.1.

The used parameters are listed in Table 6.1. Some model parameters are set independently of the measurement value in order to match the outputs of the real plant. Some values are not reached, due to model simplifications.

The fraction of the waste components, according to Bundesamt für Umwelt (Switzerland) [40] and Umweltbundesamt (Austria) [39], is shown in Table 6.2. The heating values are shown in Table 6.3.
### Table 6.1: Model parameter verification against 'KVA Turgi'

<table>
<thead>
<tr>
<th>Component</th>
<th>Parameter</th>
<th>Unit</th>
<th>KVA Turgi</th>
<th>WtE model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waste market</td>
<td>Domestic waste mass flow</td>
<td>kg/s</td>
<td>1.36</td>
<td>1.36 (mean)</td>
</tr>
<tr>
<td></td>
<td>Industrial waste mass flow</td>
<td>kg/s</td>
<td>1.36</td>
<td>1.36 (mean)</td>
</tr>
<tr>
<td></td>
<td>Sludge mass flow</td>
<td>kg/s</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>Gate fee</td>
<td>CHF/t</td>
<td>130-250</td>
<td>200</td>
</tr>
<tr>
<td>Bunker</td>
<td>Fill level</td>
<td>t</td>
<td>1.000</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Out mass flow</td>
<td>kg/s</td>
<td>4.17</td>
<td>4.17</td>
</tr>
<tr>
<td>Combustion</td>
<td>Air in mass flow</td>
<td>kg/s</td>
<td>14.60</td>
<td>7.10</td>
</tr>
<tr>
<td></td>
<td>Air in temperature</td>
<td>K</td>
<td>326.73</td>
<td>326</td>
</tr>
<tr>
<td></td>
<td>Flame temperature</td>
<td>K</td>
<td>1213</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Flue gas mass flow</td>
<td>kg/s</td>
<td>17.94</td>
<td>12.21</td>
</tr>
<tr>
<td></td>
<td>Heat energy flue gas</td>
<td>MW</td>
<td>70</td>
<td>49.61</td>
</tr>
<tr>
<td></td>
<td>Heat loss</td>
<td>MW</td>
<td>14.88</td>
<td>4.491</td>
</tr>
<tr>
<td>Water steam process</td>
<td>Steam outlet temperature</td>
<td>K</td>
<td>668</td>
<td>440</td>
</tr>
<tr>
<td></td>
<td>Steam mass flow</td>
<td>kg/s</td>
<td>15.55</td>
<td>15.55</td>
</tr>
<tr>
<td></td>
<td>Pump outlet pressure</td>
<td>bar</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Electric power out</td>
<td>MW</td>
<td>8.74</td>
<td>7.4</td>
</tr>
<tr>
<td></td>
<td>Turbine steam pressure (1st stage)</td>
<td>bar</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>District heating power out</td>
<td>MW</td>
<td>6.1</td>
<td>6.5</td>
</tr>
<tr>
<td></td>
<td>District heating network mass flow</td>
<td>kg/s</td>
<td>12.50</td>
<td>12.50</td>
</tr>
<tr>
<td></td>
<td>Cooling water mass flow</td>
<td>kg/s</td>
<td>694.44</td>
<td>694.44</td>
</tr>
<tr>
<td></td>
<td>Cooling water temperature</td>
<td>K</td>
<td>298</td>
<td>290.75</td>
</tr>
</tbody>
</table>

### System Model Analysis

In order to motivate methods for strategic control under uncertainty, the behavior of the simulation is analyzed in terms of temporal behavior (system dynamics) and uncertainty of the system reactions on control actions.

The system state is defined in three dimensions: The heating energy of the production, which represents the total energy production of the power plant ($Q_{burner}$); the heating energy that is transferred into the district heating network ($Q_{DH}$) and the electric power production $P_{EL}$. The actions are the waste mass inflow of the combustion, and three value position between 0 and 1 in the water steam cycle: The water injection at the boiler, the turbine bypass and the split value between district heating and second turbine stage.
6.3 Contribution

<table>
<thead>
<tr>
<th>Components</th>
<th>Fraction [weight-%] (mean and 2(\sigma))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Domestic waste</td>
</tr>
<tr>
<td>(H_2O)</td>
<td>22 (6.5)</td>
</tr>
<tr>
<td>(C)</td>
<td>33 (2.5)</td>
</tr>
<tr>
<td>(H)</td>
<td>4.2 (0.4)</td>
</tr>
<tr>
<td>(O)</td>
<td>19 (1.9)</td>
</tr>
<tr>
<td>(N)</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6.2: Waste fraction [40][39]

<table>
<thead>
<tr>
<th>Waste type</th>
<th>Heating value [MJ/kg]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domestic waste</td>
<td>11.4</td>
</tr>
<tr>
<td>Industrial waste</td>
<td>14</td>
</tr>
<tr>
<td>Sewage sludge</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 6.3: Waste heating value [40][39]

Due to the thermodynamic behavior of the power plant, in particular the heat exchanger and the combustion process, it is assumed, that changing control actions will effect the system state with delay. This delay is caused by the delayed reaction on input changes of the components in the water steam cycle.

Before the system dynamics of the water steam process is analyzed, the dynamics of a single NTU (Number of transfer) heat exchanger is demonstrated, which is part of the Thermolib-Toolbox. The component exchanges heat between two mass flows. If the energy level of one flow is changed, the effect on the second flow is assumed to be delayed due to the thermal mass of the component.

The mass flow of both water flows is set to 1 kg/s with a pressure of 5 bar and an initial temperature of 298 K. At time 100 s, the first inflow is heated up between 10 and 20 MW. The resulting changes of the outlet temperature of the second flow is shown in Figure 6.13(a). As expected, the outlet temperature increase with delay, due to the thermal mass of the heat exchanger.

Figure 6.13(b) shows the temperature response (outlet of the second flow) of the component of a heat injection if 20 MW for different thermal masses between 0.1254 and 0.3762 MJ/kg. It shows, that the reaction is slower if the component has more thermal mass. This is reasonable, since it takes more time to heat up the increased thermal mass.

Based on the analysis of the heat exchanger component, which is part of the water steam cycle of the energy plant, the following section shows the responses of the overall water steam process. In particular, the effect of control actions (valve positions) onto the electric power and district heating power production.

Uncertainty is given by the inhomogeneous waste composition, which effects the input energy of the water steam process. In order to show the system dynamics of the components of the water steam process without noise, the waste composition is and the mass flow of the combustion is stabilized.

Figure 6.14(a) shows the energy production for a district heating split step from 40 to 30 % (turbine
bypass 20 and water injection 0) at time 100 s. By feeding more steam into the 2nd turbine stage, the production of electric energy increases. The heating production drops. However, the drop of the heating production shows a higher delay than the increase of the electric energy production. The quicker reaction of the turbine is reasonable due to the thermal mass of the heat exchanger. The overall energy production drops from 12.2 to 11.8 MJ/s, which stems from the different efficiency levels of turbine and heat exchanger. In summary, the district heating split valve allows switching the energy production between district heating and electric energy output with respect to thermodynamic delay and the efficiency factors of the components.

Figure 6.14(b) shows a step response of the turbine bypass value onto the energy production. The bypass value opens from 0 to 20 % at time 200 s (district heating split is 50 and water injection is 0 %). As a result, both the electric energy and the heating energy production drops from 6.8 and 7.25 MJ/s to 5.6 and 6.8 MJ/s and the overall energy production is decreased. This is reasonable, since the valve lets the superheated steam bypass the turbine and the district heating, and is eventually relaxed by the plant condenser.

Figure 6.14(c) shows the step response of injecting liquid water to the superheated steam in order to cool it down. The injection value is open from 0 to 20 % at time 200 s. The turbine bypass is set to 0.0 and the district heating split is set to 0.5. The district heating energy production is sparsely dropped from 7.24 to 7.17 MJ/s and the electric energy production is dropped from 6.9 to 6.4 MJ/s. In contrast to the step response of the bypass value, the energy production does not show any dynamic behavior.

This section has shown the system reaction on control actions in terms of step responses. To connect the Reinforcement learning controller to the system model, the I/O interface must be defined in order to bridge the gap between the continuous simulation variables and the discrete states and action of the
reinforcement learners and MDP controllers. The I/O interface is developed in the next section.

### 6.3.3 Problem characteristics

Due to the multiple objectives of physical limitations of the plant and the multiple energy markets (district heating, electricity market), the given problem is an optimal control problem. In order to justify applying POMDPs to the given problem, it must fulfill the criteria listed in Section 4.1.

The given problem is a control problem under uncertainty. The controlled system is uncertain due to the inhomogeneity of the waste, which leads to unstable combustion and heat production. The random waste composition will be discussed in detail in Section 6.3.2. The dynamics of the system are the thermodynamic behavior of the components in the water-steam cycle and the latency of combustion outlet heating energy. This aspect will be discussed in detail in Section 6.3.2.

The cost function (reward model) expresses the changing market conditions, such as demand and price, as well as the gate fee income from purchasing waste and penalties otherwise. It also represents the reward which the plant gets for producing energy and purchasing waste. The selling market conditions are represented as demand and price and the waste market is represented as amount and gate fee income from purchasing waste and penalties otherwise.

The system states are the physical states of the plant, which is the mass flow, the temperature, the pressure and energy of the substance waste and water. The energy output of the plant, in terms of district heating and electric energy, is extracted from the energy in the water steam cycle which is fed from the combustion flue gas.

Observations are measurement values of the physical states of the plant.

Managing power plans entails making many different decisions (actions). Three main decision points are considered, as shown in Figure 6.15. Firstly decisions can, to a certain extent, be made at the waste purchase. Although plants commonly guarantee accepting domestic waste, they have more flexibility regarding commercial and industrial waste. An example is clearing and sewage sludge, biosolids that WtE plant managers may choose to decline or accept depending on current combustion plans. The purchase of industrial waste controls the heating value of the plant’s fuel.

The second decision point is the bunker outlet. The bunker provides a buffer between incoming waste and combustion, allowing the plant manager to make decisions regarding combustion rates and waste composition. Since different waste types burn at different speeds and produce differing amounts of heat energy, the decision of what to burn and when to burn it has a direct impact on overall energy production.
output.

And thirdly, decisions are made regarding the distribution of the produced energy. The split factor defines how much energy is put into the district heating system and how much is transformed and sold on the electricity market. The energy split is controlled by two values: The bypass value of the turbine, which channels the steam directly to the condenser and the extraction between the two turbine stages, which channels the steam directly to the district heating exchanger instead of impelling the second stage of the turbine.

Controlling the set points of the plant is a sequential decision process since the ranges of the set points are physically limited. The energy changes of the water in the steam cycle immediately effect the turbine and the electric energy production. Due to the thermal mass of the component, the effect onto the heating exchangers and the district heating production is, however, delayed.

The Markov property is assumed to be fulfilled due to the quasi-stationary simulation of the plant. Although this is not true for real physical systems, the simulation model is already simplified due to computational effort.

The time horizon is assumed to be infinite. And the state and action space discretization is defined manually in Section 6.3.4.

Simplification

Taking into account the uncertainty of observations is realistic, but it adds complexity to the model. Commonly, a measurement model is introduced to cover measurement uncertainty in terms of sensor calibration or discretization. Agueda et al. present a manually derived observation model for the drum level control in the boiler [34]. The added value of manually defined models is debatable. Considering a measurement model, however, increases the computational complexity of the problem. If the measurement model covers only general sensor errors in power plants, it is reasonable to either replace the sensor or extend the sensor unit with an estimator instead of considering such uncertainty in the planning process.

Hence, the power plant optimization will be considered as MDP problem.

The incineration plant is implemented in MATLAB/Simulink®. However, this complex model causes high computational effort and simulation times. Hence, the plant model is simplified to the water steam cycle with a flue gas input that is fed by a random generator, according to the output of the combustion block. The following sections are restricted to this simplification.

6.3.4 Automated policy deduction

MDPs are a useful construct for intelligent decision making under uncertainty. Planning algorithms automatically deduce strategic policies from a given model.

In Section 4.2, several methods for automatically deducing models and strategic control policies are discussed: They are automated planning (policy deduction) from models that are a) manually defined, b) passively deduced from measurement data, c) actively extracted from a simulation or the policy is directly learned.

The manual definition of transition probabilities, observation probabilities and rewards for systems with large state-, observation- and actions-spaces as well as complex reward models is however rarely possible
Automatically deducing the (PO)MDP model is investigated by Stollmann [204]. The controlled system is modeled using intuitive graphical tools, such as \textit{MATLAB/Simulink®}, and automatically transferred into a POMDP model, which is then used by a planner to deduce a control policy. This approach is arbitrarily challenging, due to the computational complexity of brute-force simulation. While the simplicity of this approach has merit, it remains largely intractable for real-world systems.

To overcome this issue, reinforcement learning (see Section 2.6) provides a framework for both building models and setting up an optimal control policy automatically over time.

Statistical learning methods [184], like the EM algorithm [71], deduce models from measurement data, which are then used to plan a control policy. However, statically defined models, even if automatically built, are not able to correctly represent the system dynamics.

Instead of deducing a model, another approach is permitting a controlling agent to interact with a simulated plant and learn optimal control policies from this interaction. Such reinforcement learning agents, however, do not rely on a previously defined model but rather learn the effect (value) of applying an action in a certain system state from interaction with a real system or simulation. This approach saves the effort for deducing the policy by planning on an imperfect model approximation. In order to interact with the system, an interface is required that supports the immediate reward and the next state of the system (output) as a result of applying an action (input), as shown in Figure 6.16.

Common implementations of reinforcement learning are introduced in Section 2.6. In principle, the agent applies actions to the system and observes the resulting reward and state. \textit{Q-learning} [184] approximates the value function (see Section 2.4.3). It memorizes an average of the knowledge, i.e. q-value, to deal with the system’s uncertainty. In any case, in a setting where an agent is learning whilst interacting with a system, a dilemma quickly appears: As the agent’s understanding of the system increases, a choice has to be made about whether to \textit{exploit} the existing knowledge or attain further knowledge through \textit{exploration}, i.e. looking for potentially greater profits later in time.

The selected strategy of exploit vs. exploration has enormous effects on a learner’s control quality. Performing the most rewarding actions within the existing knowledge may not get the system into more valuable states while randomly exploring the system might lead to undesired states. In general, control with partial knowledge cannot guarantee optimal control, whereas infinite exploration is not feasible. Convergence proofs regarding reinforcement learning algorithms such as q-learning assume that every
state is visited an infinite number of times and every action is performed an infinite number of times from everyone of those states [216]. However, finding good learning parameters to quickly converge the approximated policy towards the optimal policy without getting stuck due to the high effort of exploring the system behavior is a challenging task, which depends on the structure of the specific problem.

Model-based learning, however, combines learning the value function and the model with planning in order to optimize learning and active control. In contrast to common reinforcement learning, model-based reinforcement learning additionally memorizes the system dynamics. It explores this already gathered knowledge using planning in order to improve the learning strategy.

In general, reinforcement learning can deal with changing environments such as changing market rewards. To apply learning getting an optimal control policy for changing market situations, the plant can be modeled in any language that supports an interface for the learner software.

In the following, the methods of model-free reinforcement learning are applied to the optimization of a large real-world problem, in particular a power plant.

I/O Interface: Quantization of Continuous Model

The I/O interface is implemented as part of the software package pyMDP, which is presented in Appendix D. This novel software connects simulation in MATLAB/Simulink® and controllers in order to combine AI software written in the well-equipped programming environment Python, which for example allows to easily integrate program code written in C, and MATLAB/Simulink®, which allows to easily set up simulations.

MDP controllers and reinforcement learning agents, as introduced in Section 2.4, 2.6 and 6.3.4, control a system by discrete interactions over discrete time. Such interactions occur sequentially and fulfill the Markov property, which is depending only on the current (discrete) system state.

In contrast, Simulink models of physical systems are usually defined as differential equations over continuous spaces and variable step-step solvers, such as ode45, may develop the system dynamics at arbitrary points in time. Moreover, the learner represents states and actions of the system as no actual values but rather a simple index. This poses a problem in mapping system state and actions between the simulation and the learner.

The issue of arbitrarily discrete time, due to the solver, can be approached in two principle ways [205]. A first approach is to use a fixed-step solver, which guarantees the availability of output values at the required time. The main disadvantage of using a fixed-step solver is that numerical errors cannot be mitigated by increasing or decreasing the step-size depending on the system dynamics. It is thus impossible to guarantee that the solver will remain within defined tolerance levels. The second approach is to ensure that the maximum step-size remains small enough to be inconsequential with long simulation times. A further reason why this approach is applicable to the given problem is the stochastic nature of the model, which will inherently lead to small variations that are likely to be much larger than those introduced by a variable step-size (assuming the minimum step-size is chosen appropriately).

Actions are the value positions for the steam cooling injection (at the boiler), the turbine bypass and the split between the second turbine stage and the district heating exchanger (DH split). An action, in the sense of the controller, represents one element of the Cartesian product of variables that are listed in table 6.5. The values are selected according to physical limitations.

The valve positions are defined in percent. 0 means a fully closed valve. The open district heating split valve means that all steam from the first turbine stage is fed into the district heating exchanger. The
range of the valve is set around equally splitting the steam flow onto the second turbine stage and the heat exchanger. A fully open turbine bypass valve feeds the steam directly to the condenser, bypassing the turbine. Hence, the range is set between 0 and 20 in order to mainly feed the turbine. The valve for steam cooling injection injects no liquid water if the valve is not open. It is reasonable to allow only little injection up to 10 percent in order to not destroy the turbine due to wet steam.

The action mapping, listed in Table 6.5, results in 75 actions. Action 23, for example, represents \{DH split = 40, Bypass = 10, Steam cooling injection = 5\}.

Listing 6.1 shows the implementation. Mapping actions to values in the model is done with a MATLAB struct array, which is initialized when the model is loaded. Figure D.2(a) shows the Simulink block which is responsible for translating the action index, given by the Simulation request, to the correct values of the decision variables of the Simulink model. (The values in Table 6.5 are mapped onto the interval \([0,1]\) for technical reasons.)

**Listing 6.1: MATLAB Action Mapping**

```matlab
%% Action mapping
%
%% Valve Position 1: For Split on District Heating [0.3,0.4,0.5,0.6,0.7]
%% Valve Position 2: For Turbine Bypass on Limmat [0,.05,.10,.15,.20]
%% Valve Position 3: Steam Cooling Injector [0,.05,.1]
%
%% Note: the learner actions are indexed starting from 0, whereas Matlab
%% only permits indexing from 1, so a lookup shoudl be done like:
%% ActionMapping(<action_id>+1)
%
%% Note: total number of acitons is 75
ActionMapping = struct('v_pos_1',{},'v_pos_2',{},'v_pos_3',{});

%% Action Definition
ind=1;
for f=0.3:0.1:0.7
    %Iteration Valve 1: DH-Split
    for k=0:0.05:0.2
        %Iteration Valve 2: TG-Bypass
        for i=0:0.05:0.1
            %Iteration Valve 3: Injector
            $\text{Actions}$
            ActionMapping(ind).v_pos_1 = f;
            ActionMapping(ind).v_pos_2 = k;
            ActionMapping(ind).v_pos_3 = i;
            ind=ind+1;
        end
    end
end
```

### Table 6.4: State discretization

<table>
<thead>
<tr>
<th>Variable</th>
<th>Values [MW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\dot{Q}_{\text{Burner}}$</td>
<td>30, 40, 50, 60, 70</td>
</tr>
<tr>
<td>$\dot{Q}_{\text{DH}}$</td>
<td>6, 6.5, 7, 7.5, 8</td>
</tr>
<tr>
<td>$P_{\text{El}}$</td>
<td>5, 5.5, 6, 6.5, 7, 7.5, 8</td>
</tr>
</tbody>
</table>

### Table 6.5: Action discretization

<table>
<thead>
<tr>
<th>Variable</th>
<th>Values [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>DH split</td>
<td>30, 40, 50, 60, 70</td>
</tr>
<tr>
<td>Bypass</td>
<td>0, 5, 10, 15, 20</td>
</tr>
<tr>
<td>Steam cooling injection</td>
<td>0, 5, 10</td>
</tr>
</tbody>
</table>

### Table 6.6: I/O interface and model discretization
The second aspect of the I/O mapping is the state mapping. The dimensions (or variables) of the state space are the produced heating energy of the combustion ($\dot{Q}_{Burner}$) in MW, the heating energy that is fed into the district heating ($\dot{Q}_{DH}$) in MW and the electrical power production $P_{El}$ in MW. Table 6.4 lists the state mapping. A state, in the sense of the controller, represents one element of the cross product of variables. The cross product of the state variables leads to 175 states. State 50, for example, represents the element {\(\dot{Q}_{Burner} = 50\), \(\dot{Q}_{DH} = 7\), \(P_{El} = 5\)}.

The implementation of the state mapping is shown in Listing 6.2.

### Listing 6.2: MATLAB State Mapping

```matlab
%% State mapping
% State variables:
% - Qdot_Burner: {30,40,50,60,70}
% - Qdot_DH: {6,6.5,7,7.5,8}
% - P_El: {5,5.5,6,6.5,7,7.5,8}
% 
% Note: the learner states are indexed starting from 0, so the index returned
% to the learner should be the one returned from findState, substracted
% by 1.
% 
% Note: total number of states is 175
StateMapping = struct('Qdot_Burner',{},{},'Qdot_DH',{},{},'P_El',{});

%% State Definition
Qdot_Burner_s = [30,40,50,60,70];
Qdot_DH_s = [6,6.5,7,7.5,8];
P_El_s = [5,5.5,6,6.5,7,7.5,8];

n = 0;
for n1=1:5
    for n2=1:5
        for n3=1:7
            n = n + 1;
            StateMapping(n).Qdot_Burner = Qdot_Burner_s(n1);
            StateMapping(n).Qdot_DH = Qdot_DH_s(n2);
            StateMapping(n).P_El = P_El_s(n3);
        end
    end
end

% Function to find a state ID given a vector of four values, consecutively
% Energy_Generator, Energy_DH, Energy_Burner
findState = @(vs) find(arrayfun(@(x) all(x.Qdot_Burner == vs(1)) && x.Qdot_DH == vs(2) && x.P_El == vs(3)), StateMapping));

findStateCantFail = @(vs) find([[1]],arrayfun(@(x) all(x.Qdot_Burner == vs(1)) && x.Qdot_DH == vs(2) && x.P_El == vs(3)), StateMapping)),1,’last’);
```
6.3 Contribution

Figure 6.17: Q-value history for state 74 with $\alpha = 0.5, \gamma = 0.5$

Results

The purpose of running a learner on a simulation model is deducing a control policy. A value function stores the value for state, action tuples representing the value according of applying an action to a certain state. Then, the (optimal) control policy is given by the action with the best value.

As discussed in Section 6.3.4 and 2.6, q-learning develops such a q-value function $Q(s_t, a)$ for state $s_t$ and action $a$ according to Equation 2.32. It gets the reward $R(s_t, a)$ and the new state (transition) from sampling the system simulation. The desired q-value development over time is a converging value for each action and a best action for each state under uncertain, strategic effects.

As introduced in Section 2.6, q-learning is parametrized by three parameters: The learning rate $\alpha$, the discount factor $\gamma$ and, in case of a $\epsilon$-greedy learning policy, by the greediness factor $\epsilon$.

In the following, the influence of the learning parameter setup $x$, shown in Equation 6.1, is investigated in terms of exemplary q-value developments over the learning period of 500 learning epochs with a length of 10 s each. The q-value development shows the value development for (state,action) tuples over the learning process time. The resulting policy is given by the action with the highest value at the end of the learning process. This is indicated by the graph with the highest value at the end of the learning time.

$$x = (\alpha, \gamma, \epsilon) \in \{(0.5, 0.5), (0.7, 0.7), (0.9, 0.9)\} \times \{0.5, 0.7\}$$  \hspace{1cm} (6.1)$$

Figure 6.17 shows the influence of the learning parameter $\epsilon$ onto the q-value development for state 74 ($\{\dot{Q}_{Burner} = 50, \dot{Q}_{DH} = 6, P_{El} = 6.5\}$) with $\alpha = 0.5, \gamma = 0.5$. Figure 6.17(a) indicates that the learner found action 7 ($\{DH \text{ split} = 30, Bypass = 5, \text{ Steam cooling injection} = 10\}$) to be best. As shown in Figure 6.17(b), the learner comes to a slightly different policy with $\epsilon = 0.5$. In this case, action 9 ($\{DH \text{ split} = 30, Bypass = 10, \text{ Steam cooling injection} = 10\}$) is clearly preferred, due to the high q-value. Figure 6.17(c) shows the value development for a greediness of $\epsilon = 0.7$. In this case, the policy selects action 7 ($\{DH \text{ split} = 30, Bypass = 10, \text{ Steam cooling injection} = 0\}$) to be best in state 74. As a result, increased greediness shows an instable value development over time for state 74.

Figure 6.18 shows the influence of the learning parameter $\epsilon$ onto the q-value development for state 74 with $\alpha = 0.7, \gamma = 0.7$. It shows that the best action differs for the three learner setups, although action 17 ($\{DH \text{ split} = 40, Bypass = 0, \text{ Steam cooling injection} = 5\}$) is dominating mostly for $\epsilon \in \{0.5, 0.7\}$. For $\epsilon = 0.3$, the learner deduces action 21 ($\{DH \text{ split} = 40, Bypass = 5, \text{ Steam cooling injection} = 10\}$) to be best. With $\epsilon = 0.5$ it results in action 17, and with $\epsilon = 0.7$ it concludes that not action 17 but eventually action 11 ($\{DH \text{ split} = 30, Bypass = 15, \text{ Steam cooling injection} = 15\}$) is best.
Figure 6.18: Q-value history for state 74 with $\alpha = 0.7$, $\gamma = 0.7$

Figure 6.19: Q-value history for state 74 with $\alpha = 0.9$, $\gamma = 0.9$

Figure 6.19 shows the influence of $\epsilon$ onto the q-values for state 74 with $\alpha = 0.9$, $\gamma = 0.9$.

For $\epsilon = 0.3$, action 22 ($\{\text{DH split} = 40, \text{Bypass} = 10, \text{Steam cooling injection} = 0\}$) is eventually dominating. For $\epsilon = 0.3$ and 0.5 it is consistently action 2 ($\{\text{DH split} = 30, \text{Bypass} = 0, \text{Steam cooling injection} = 5\}$).

However, in any setup no clear convergence is shown for any setup within the 500 learning epochs. Extending the learning phase is necessary to investigate the convergence of the q-values in detail. This make, however, high effort due to the simulation runtime of the given problem and the high number of samples.

**Computational Complexity and Runtime**

Simulating all states-action combinations for the given example one time results in $13.125 \times 175 \times 75$ simulation runs. This does, however, not cover the probabilistic system dynamics realistically, since each combination should be sampled multiple times. A single simulation runs with a real-time/simulated-time factor of approximately 3. This means, that running 13.125 simulations with an epoch length of 60 s takes 3 days, without covering the probabilistic system behavior. Sampling each state and action tuple is, however, not enough to consider the long term effects of an action. Nevertheless, staying in a subset of states increases the number of visits per state for a given learning period.
6.3.5 Policy evaluation

Reinforcement learning gathers the value of a state-action tuple from a dynamic system by sampling its behavior and reward. The action with the highest q-value defines the best choice for the corresponding state. This represents the learned policy that differs in quality. Evaluating the policy is investigating the quality of the policy.

The policies differ in several aspects controllable and uncontrollable aspects. Since the policies are deduced from a probabilistic simulation, the policy is influenced by the underlying random process, which is not controllable. Controllable aspects are, first, the learning parameters, which are for the q-value function $\alpha$ and $\gamma$. Second, the strategy of the learner, which is been parametrized by $\epsilon$, in the previous chapter. And finally, the learning horizon, which defines how many samples the learner takes from the system.

A reasonable indicator for the quality of learning is the convergence of the q-value over the learning time. However, it strongly depends on the learning rate. If the learning rate is set to zero, the value obviously converges without any gained information. As shown in the last section, The q-values do not clearly converge to an optimal value. Nevertheless, not the q-value but the resulting policy is important to be evaluated.

Instead of investigating the q-value development over time, the resulting policy can be evaluated independently of the learner internals. A controller can apply the learned policy to the controlled system or simulation. Investigating the accumulated reward which the controller gets over time, allows to compare the controller, the policy and the learner, respectively.

This section compares the different learner setups from the previous section by evaluating the resulting policies on the simulation model.

6.3.6 Setup

To evaluated the policy, a controller loads the learned policy $\pi(S) \rightarrow A$ in terms of $\pi(s) = \arg \max_{a \in A} \{Q(s, a)\}$ and applies it to the simulation for a certain time frame. Like the learner, the controller are connected to the simulation via the pySim interfaces described in Section 6.3.4. Like the learner, shown in Figure 6.16, the controller applies an action to the simulation and get a reward and the next system state in time from the simulation. In contrast to the learner, the controller only applies a static policy, which is defined according the system state $s$, as already described.

If a controller with policy $\pi_A$ earns more total reward than a controller with policy $\pi_B$, $\pi_A$ is better than $\pi_B$ w.r.t. the taken sample. And therefore the leaner setup for getting $\pi_A$ is better for the given problem.

6.3.7 Results

Figure 6.20 shows the accumulated rewards for 9 controllers being applied to the KVA simulation. The controller with the most gathered reward, performs best. Thus, the underlying policy is best and the leaner setup, which gathered the policy is best for the given problem.

All policies show a positive development, which represents that the controllers earn reward. They differ in the gradient, which results in different total rewards after 5000 s of evaluation.
For the first setting \((\alpha = 0.5, \gamma = 0.5, \epsilon \in \{0.3, 0.5, 0.7\})\), shown in Figure 6.20, the influence of the greediness onto the reward results in slightly differing gradients. The three learner setups result in policies with similar rewards. After 5000 s the highest accumulated reward of 2695 is gained by the setup with \(\epsilon = 0.5\).

The reward for the second group of learner setups \((\alpha = 0.7, \gamma = 0.7, \epsilon \in \{0.3, 0.5, 0.7\})\) does not show similar behavior. The setups clearly differ in the gradient. In this case, the greediness has a much higher effect onto the quality of the resulting policy. The setup with \(\epsilon = 0.5\) clearly performs best within the second group and outperforms also all other setups.

The third setup \((\alpha = 0.9, \gamma = 0.9, \epsilon \in \{0.3, 0.5, 0.7\})\) is fully dominated by other policies over time. It shows the worst performance. The difference between the setups in this group are similar to the second group. The middle epsilon performs worst.

In conclusion, the policy deduced by the learner setup \((\alpha = 0.7, \gamma = 0.7, \epsilon = 0.5)\) performs best.

It is intuitively reasonable to compare the quality of the learner setups not by the development of the q-value function during the learning period, but by the accumulated reward of actually applying the resulting policy to the system.

However, sampling the controller behavior does not necessarily completely cover the operation space nor the system behavior. It is still just a sample. Nevertheless, it is a reasonable approach to compare learners, if the sample is representative.

### 6.4 Summary

This chapter shows how to deduce strategic control policies from a simulation of a complex system, i.e. a thermal power plant. However, the presented results are a prove of concept of the chain between manually modeling the problem and automatically deducing as well as evaluating policies.

The presented software closes the gap between manually modeling the problem and automatically deducing as well as evaluating policies. The prove of concept establish the theoretical approach, which is presented in Section 4.2.2.
In particular, this chapter describes how to apply a Markov decision process learner to power plant control. The introduction into power plants shows today’s difficulties in energy production. First, the demand for electric energy rapidly changes on hourly bases. As a result of the liberalization of the electric energy market, this also affects the price. Second, the energy production of the emerging regenerative power plants, such as wind parks, strongly depends on the weather. To keep the electricity grid stable, thermal plants are capable to compensate these issues. However, optimally operating the plant from an economical perspective with respect to the physical dynamics of the plant, is a challenging task.

Power plant are dynamic systems. The thermodynamic behavior of thermal power plant requires considering the system dynamics by a predictive controller, in order to optimally control the energy production. Uncertainty is given due to the fuel of the waste to energy plants. Since waste to energy plants burn waste to produce energy, the combustion process is instable due to the inhomogeneous composition of the waste. Delivering not only electric energy but also heating energy to the local district heating network or process steam to industry is common for thermal power plants. This makes the problem of economically optimal energy production being a multi-objective control problem. To adapt the energy production to the current market situation (demand and price), the operator can change the combustion set point and the energy split, which defines the amount of the overall energy production that is fed into the electric grid or the district heating.

The previous chapter contributes to applying strategic control under uncertainty to large, real world problems. Sequential decision-making, such as Markov decision processes, provides a framework for controllers that decide according to strategic policies under uncertainty.

Complex system models are hard to express as explicit Markov decision process, in terms of probability matrices or Bayesian networks. However, reinforcement learning automatically deduces a policy with respect to a reward function from an arbitrary system. Expressing the complex system as simulation allows to automatically deduce policy by learning. This allows to separate modeling and optimal control.

To ease the development of MDP controller and learner for industrial applications, a toolbox is developed in MATLAB/Simulink® and Python. It extends the existing software providing controller, learner and planning algorithms as well as the underlying data structures for applying, learning, calculating, evaluating and representing MDP problems in Python. Furthermore, it connects MATLAB/Simulink® simulations with the controller and learner in order to evaluate existing control policies and learn control policies from complex models. Being modeled in the high-level language MATLAB/Simulink®, process engineers can model complex systems.

Hence, learner sample simulations in order to deduce strategic control policies and controllers are applied to system simulations to be evaluated.

The software is applied to strategic energy production of a waste to energy plant. The model represents uncertain dynamics of the controlled system, in particular a waste-to-energy plant, and the economic aspects in terms of price and demand models for energy production and waste. In order to express a realistic model, the model structure and parameterization is deduced from data of the waste to energy plant KVA Turgi. Then, the model is simplified in order to apply reinforcement learning. The I/O interface discretized state and action spaces and connects learner, controller and the system simulation. To evaluate policies, different learner setups are applied to the system simulation and result in different control polices, which are discussed in detail. Finally, the policies are compared by capturing the gained reward of applying them to the simulation model.
6 Case Study: Strategic power plant production control

6.5 Outlook

Although the system model already describes many aspects of the power plant, the control method forces to simplify the model in order to reduce the computational effort of learning.

In order to represent the given problem more accurate, the model can be extended as follows. The bunker can be extended as composition of multiple piles. Each one optimally mixed, which smooth the waste composition. Adding a decision variable to select the pile (inlet and outlet) allows controlling the waste composition of the bunker piles (inlet) and the outlet of the bunker that feeds the combustion. Controlling the waste composition of the combustion effects the overall energy production in terms of the combustion heating value in the long run.

To control the overall energy production of the plant, an additional decision variable could control the target temperature of the combustion. The combustion process can be controlled by a PI controller in terms of input mass flow and temperature of the air input. Target of the controller are the air excess, carbon fraction of the slag and the combustion temperature, respectively the flue gas temperature, which should be at least 1123 K, to assure thermal destruction of dioxins according to Bardi et al. [45].

However, the presented results are a proof of concept of the complete chain between manually modeling the problem and automatically deducing as well as evaluating policies.
Conclusion

This thesis contributes to applying strategic decision making under uncertainty to real world problems by presenting extensions to the current state of the art. They are developed in principle and applied to concrete problems.

In Chapter 4.1 are the required characteristics that justify the application of POMDPs listed and discussed.

How to deduce strategic control policies under uncertainty for real world problems, is answered theoretically and shown practically.

Section 4.2 describes the dependence of the model components and discusses several modeling processes. Commonly, POMDP models are defined manually. This might lead to modeling errors. In order to define the problem realistically, the first contribution of this thesis is deducing POMDP models from measurement data. This reduces the number of free model parameters, which are usually defined manually. The modeling process from measurement is defined and applied to the concrete problem reducing the parallax error on interactive screen.

It is found that the parallax error depends on the users viewpoint, which is not directly measurable. Therefore, instead of measuring the viewpoint, the presented parallax correction controller is designed to adapt the error correction according to interaction events on the touch screen. It is found that the correlation contains uncertainty. It is, moreover, shown that the user moves in front of the screen while interacting with the system. Hence, a static correction cannot overcome the parallax error. To continuously adapt the correction with respect to uncertainty, it is formulated stepwise. In particular, it is expressed as POMDP. The underlying model is deduced from measurement data of studying the user’s behavior. The study provides observation and the transition model reducing the model aspects, which must be defined manually, to the reward model. In order to decrease the effort of planning the control strategy, the model is then ingeniously reduced in size (small problem definition). Developing the parallax error correction controller raises the question of how to deal with observation events, since interaction events occur at varying times.

The second contribution is an oblivion process for Bayesian filtering. Commonly, controllers gather sensor information at each control loop step to update the estimate of the system state and apply optimal control actions appropriately. Event-based observations, however, do not provide information at each control step. For this case, an oblivion process is proposed: If no event occurred, an oblivion step develops the estimate stepwise towards the uniform belief state, which represents no knowledge.

The third contribution focuses on large problems. It answers the question of how to model large problems in order to apply sequential decision making under uncertainty.

Real world problems, such as power plants, are large and complex. Applying POMDP is hardly possible
as manual modeling probability matrices or Bayesian networks is unclear, imprecise and causes errors. This directly affects the resulting control policy. However, modeling such problems is possible in a high level language, such as MATLAB/Simulink. Providing a graphical user interface for modeling dynamic system with differential equations, the software allows process engineers to effectively express the given problems.

In order to apply sequential decision making under uncertainty to such problems, a software framework is developed. The software automatically deduces strategic control laws from MATLAB/Simulink simulations using model free reinforcement learning. It extends the existing MATLAB/Simulink toolboxes by MDP, POMDP and reinforcement learning algorithms in order to apply sequential decision making under uncertainty to MATLAB/Simulink simulations. In contrast to most of the existing work, this approach assures that the control policy is deduced from a valid model because Simulink models can be implemented by process experts, adapted to real systems and also been verified with measurement data. The software is applied to a power plant simulation to develop an optimal control policy for demand driven energy production.

This work closes the gap between sequential decision making and large real world problems. It shows how to benefit from complex control methods in industrial applications.

The proposed oblivion process has mathematically proven to be correct. The presented development process for applying sequential decision making, in particular MDPs and POMDPs, is a recommendation based on the outcomes of developing controllers for reducing the parallax error on interactive screens and demand driven energy production for power plants. The works shows, how to automatically deduce the system dynamics and measurement models. However, the topology in terms of the state space and action space, is still been defined manually. Selecting the level of abstraction of the underlying model, is still a challenging task.

The implemented software framework for connecting MATLAB/Simulink simulations with reinforcement learning and planning algorithms written in Python is a prototype. Hence, it should be further tested and redefined in future projects. To get the software to a productive and stable level, it must be further tested and fine-tuned. The inter-process communication, for example, is file-based. Compared to direct memory access, this approach is slow and, in the current implementation, not robust against race conditions, which occur if the simulations are run simultaneously to reduce the computation time. However, the software demonstrates the working principle of the proposed approach. It is the first step towards a MATLAB/Simulink software toolbox providing MDP and POMDP controllers and reinforcement learning algorithms.
Human computer interaction studies

A.1 Interaction study on a large interactive screen

In order to analyse the user behavior in terms of viewpoint location and movement, and to demonstrate the parallax error, a user study is conducted, which monitors the user interacting on a large screen with an offset of approximately 10 mm between interaction and image plane. The viewpoint of the user, the target and the interaction position are measured as three-dimensional data. The interaction error and the viewpoint relative to the target position are analyzed. It is shown that the interaction error is partly biased by the parallax error: The absolute pointing error, given by the offset between target and actual interaction position, is compared with the offset from the assumed interaction position, given on the straight line between target and viewpoint on the interaction plane.

(a) Measurement setup with test person (schematic)  (b) Assembly of the tracking system on top of the display

Figure A.1: Hardware setup of the interactive screen 'DigiBench'

A.1.1 Participants

The average age of the 13 male and 4 female participants was 31.05, their median age was 29 and the average (median) height was 1790 (1840) mm. Four test persons were left-handed and 8 wore glasses.
A Human computer interaction studies

A.1.2 Task

As shown in Figure A.1(a), the user’s task is to stand in front of the digital board and try to hit a button that appears at random positions on the screen. In [131], it is stated that targets for touch-sensitive screens should be at least 22 mm in diameter, which corresponds to the user’s fingertip. Hence, a realistic but small and symmetric target of 13 x 12 mm (15 x 15 px) is defined to motivate the user to hit the target’s center.

After each interaction, the button moves to the next position, regardless of whether the user hits the target or not. The user is encouraged to move freely in front of the screen during the task. The typical task duration (measurement acquisition time) is 5 minutes to avoid fatigue. The user’s viewpoint is not only biased by his characteristics (height and arm length), it also depends on the position of the button element on the screen.

A.1.3 Interactive system 'DigiBench'

The interactive system DigiBench is a large, touch sensitive monitor, which was developed by the ICVR research group. It is composed of a 50-inch plasma display (Pioneer PDP-502 MXE) with a resistive touch-sensitive overlay (SMART Technologies) mounted on top of the display. The display resolution of 1280 x 768 px and its effective size of 1098.2 x 620.5 mm results in a pixel pitch of 0.858 x 0.808 mm px. Each pixel is implemented as a square arranged set of cells for each primary color, with a distance of less than 0.2 mm. The front glass is divided into a front glass substrate, a dielectric layer and a protective layer to stabilize the monitor and to shield the high voltage. The thickness of the monitor front glass is assumed between 3-5 mm.

The tracking system is a touch sensitive film stretched on a 6 mm glass plane, which is mounted 2.6 mm in front of the image plane (shown in Figure A.2(b)). Hence, the overall offset between interaction plane and image plane is 10 mm. The resistive system is capable of tracking a set of passive TUIs, as well as the user’s bare finger. The tracking position is forwarded to the operating system as standard pointing device coordinates. The tracking system was statically calibrated with 6 sampling points. To avoid the influence of a changing viewpoint relative to the target, the calibration is done orthogonally by hand.
A.1 Interaction study on a large interactive screen

![System setup with subject](image1)

![3-dimensional tracking data of digibench and viewpoint](image2)

Figure A.3: Large interactive system with 3-dimensional head tracking

A.1.4 Tracking system

A *Qualisys Motion Tracking System* was used with four *Oqus 300* cameras to continuously track the position of the user’s head and of the interactive screen. Each camera emits infrared light, which is reflected by marker balls. After calibrating the tracking system with an error standard deviation of $\sigma = 0.876$ mm, the Qualisys tracking software provides 3D positions of the markers’ centers. Together with the touch sensitive overlay on the screen, the overall setup provides the 3D coordinates of the viewpoint, the interaction point and the target point relative to the upper left corner of the display (see Figure A.3(b)). The viewpoint body consists of four markers applied to glasses the user has to wear during the tests. It was also possible to wear additional correcting glasses (Figure A.2(a)). The position of the user’s viewpoint is unequivocally defined by the position of the ball, which is mounted next to his nose.

The target position as well as the touch position on the interactive screen are given as display coordinates. As will be discussed in more detail, the hit point and the target position on the screen are provided by the test application as two-dimensional display data. To compare this data with the 3D viewpoint position, the display data is transformed into 3D data. In order to map the display data to 3D spatial data, the display’s 3D position also has to be tracked. For doing so, the display is equipped with three markers in the corners (Figure A.2(b)). Since the corresponding vectors are aligned to the display dimensions, the display coordinates can easily be transformed to a global coordinate system using the display’s pixel pitch as dilation factor.

The system setup is shown in Figure A.3(a). The tracking setup covers the working area in front of the interactive display. Since the system interaction on the interactive screen is limited by the user’s arm length $a$ (between 662 and 787 (616 and 762) mm of human males (females) [73]), the tracking area is limited to $2 \cdot \max a + w'$, w.r.t. the display width $w$ times $a$ times the user’s height (between 1629 and 1841 (1510 and 1725) mm of males (females)). The tracking system captures the user’s position with an update frequency of 50 Hz.
A Human computer interaction studies

A.1.5 Data consolidation

As described before, the tracking system provides 3D data from the user’s viewpoint and the corners of the interactive screen in mm, while the test software provides the target position and the user’s click position in 2D display coordinates. Thus, this different information must be consolidated to have a single data model.

Based on the pixel pitch of the display and the 3D positions of the display corners, the logging software calculates the 3D position of the 2D display data. The display coordinates are shifted orthogonally to the image plane by the parallax offset of 10 mm with additional respect to the marker’s diameter. Given these parameters, the consolidation module queries the 3D tracking software for the markers’ 3D positions, triggered by an interaction of the user within 50 mm distance of the target to ignore accidental integrations. Then, the software records the time stamp, the calculated 3D data of viewpoint, as well as target and hit point relative to the upper left corner of the display. Logging the viewpoint, the interaction point and the target on the screen separately, the setup is robust against movement of the devices.
A.2 Interaction location on screen

The user’s viewpoint (system state) is estimated from interactions on the screen. As part of the Bayesian filter model (see Section 2.5.1), the independent prior distribution of the user’s viewpoint is deduced from measurements in Section 5.5.1. Due to the limited arm length, the viewpoint depends on the interaction position. This effects the user’s viewpoint particularly on large interactive screens. In order to simulate a common task for studying the user’s viewpoint in front of the screen, a user study is described that investigates the clicked Graphical User Interface (GUI) target positions on conventional computer systems. Therefore, the absolute two-dimensional display coordinates of pointing device interaction on desktop computers is measured during common office tasks over weeks.

The participants of the study are research assistants. The average age of the 8 participants is 39.8; the median age is 41.5. The subjects were informed about logging the pointing interactions on their workstations. The operating system on all test machines is Microsoft® Windows® XP with standard settings of the taskbar and text size. Two hardware setups were distinguished: Participants using single or equal dual monitor systems. On the dual monitor systems, the main screen is shown on the left monitor.

Figure A.4 shows the normalized results of the click position probability distribution. The data set for the single screen system shows an almost uniform distribution of clicks in the center of the screen in both dimensions, with accumulations at the upper and left part of the screen and rectangular areas on the left and lower partition with few interactions. These accumulations correspond to the common widget structure of the operating system GUI: A file-menu in the upper and a vertical scrollbar at the right area of the main application window. Moreover, in the lower screen section, the measurements show a pattern, which corresponds to the elements of the operating system’s taskbar, which is used to switch between the running programs. As shown in Figure A.4(b), the data sets for dual screen systems confirm this effect on each monitor separately, although the primary monitor is used more often.

Migge et al. [147] uses these results to simulate a common task for studying the user’s viewpoint in front of the screen, which is described in Chapter 5.5.

Figure A.4: Interaction position on screen (scatter plot and histogram)
A Human computer interaction studies

A.3 User viewpoint dynamics identification

The user’s viewpoint represents the estimated system state. As stated in Section 2.5.1, a Bayesian filter uses an observation and a process model to update the belief of the system state. As part of the observation model, the prior state distribution defines the unconditioned probability of the viewpoint $P(s)$. The process model describes the development of the system state over time as $P(s'|s, a)$, which is - in case of the user’s viewpoint - the user’s movement within a certain time step.

In this section, two characteristics of the user’s behavior are analyzed: The position and the movement in front of the screen for discrete time steps. The question about the user’s typical position and where he will probably move in the next time-step $t_{i+1}$ relative to the current position at $t_i$ should be answered. To set up a model that matches the user’s behavior closely, a user study is performed, which tracks the position of the user’s head (viewpoint) in front of a large digital whiteboard. Since the viewpoint depends on the position of the aimed widget element (e.g. GUI button) on the screen, the task for this study is set up based on empirical results of typical click positions for common everyday jobs in window applications derived from the preliminary user study.

A.3.1 Subjects

The average age of the 12 male and 2 female participants is 32.5. Their median age is 30. The average (medium) height is 179 (183) mm. Two test persons were left-handed and 1 of the test candidates wore glasses. The participants were informed about how to interact with the hardware system and the task, which is described below. They were also informed about tracking their heads during performing the given task.

A.3.2 Setup

The user study was performed while standing in front of a vertically mounted 65” interactive surface with an offset of approximately 7 mm between image and interaction plane and located 950 mm above the ground. The system is equipped with a flat panel overlay from SMART Tech. to track user inputs. It is mounted on top of the LC-display. The system is capable of tracking a set of pens and an eraser, as well as the user’s finger. The tracking position is forwarded to the operating system as pointing device coordinates. Hence, the overlay can be used to control any GUI application.

To track the user’s head continuously, a low-cost vision based 3D tracking system was developed [92]. Two Nintendo Wii Remotes [129] track a set of active IR-markers, which are attached to the user’s head. As shown in Figure A.5(a), the 3D position of the user’s viewpoint is triangulated, combining separate information of detecting the user on a straight line that goes through the position of the camera of the Wii Remote and the camera’s image plane. The tracking setup covers the working area in front of the interactive display. Since the system interaction on the interactive screen is limited by the user’s arm length (95% of human men (women) between 662 and 787 (616 and 762) mm [73, 74]), the tracking area is limited to two times the maximum arm length plus the display width times the arm length (distance to the display) times the user’s height (95 % of men (women) between 1629 and 1841 (1510 and 1725) mm). The tracking system captures the user’s position continuously with an update frequency of approximately 10 Hz.

The test application representing the user’s task (see Figure A.6 ) is implemented in Python using the Qt GUI Library [16] on a Linux system running an X-Server.
A.3 User viewpoint dynamics identification

![Tracking setup (schematic)](image1)

(a) Tracking setup (schematic)  
(b) Head tracking trace in front of the interactive screen  
(3-dimensional position in mm)

Figure A.5: 3-dimensional head tracking

![GUI application simulating common targets on the screen (screenshot)](image2)

Figure A.6: GUI application simulating common targets on the screen (screenshot)

A.3.3 Procedure

The participant’s task is to stand in front of the digital whiteboard, trying to hit a button that continuously appears on different positions on the screen. After hitting the button, the button changes its position and the process starts over. If the user misses the button, nothing happens. In the process, the user is told to move freely in front of the screen while looking at the screen. The typical task duration (measurement acquisition time) is 5 minutes. The user’s viewpoint is not only biased by his characteristics (height and arm length), it also depends on the position of the actually aimed widget element on the screen. And these targets are primarily defined by the application specific dialog design. To cover a large field of applications, a full screen GUI application was implemented that simulates common office tasks by sampling a sequence of targets from the prior defined transition probability distribution of the positions on the screen. The distribution is derived from a long-term study, logging the position of pointing device clicks on desktop computers in an office environment (see Appendix A.2 ). Hence, the resulting parallax error controller adapts best to daily business office tasks. The function is expressed in a matrix: Each row defines the transition probability from a source to all possible sink partitions of the click transition.

A.3.4 Measurement results

In this section, two characteristics of the user’s behavior are discussed. First, the measurements of the user’s absolute position is analyzed. Then, the relative movement of the user for different time intervals is investigated. The dimensions are referred as follows: The x-dimension is the horizontal dimension parallel to the surface of the screen, increasing to the right; the y-dimension is the vertical axis parallel to the screen increasing to the floor; and the z-dimension is defined to be orthogonal to the screen, increasing...
A Human computer interaction studies

Figure A.7: User viewpoint location [mm] (histogram)

Figure A.8: User viewpoint movement with 0.2 Hz [mm] (histogram)

with the distance to the screen. The origin is defined to be at the upper left corner of the display. The screen size is 1480 x 800 mm.

Figure A.5(b) shows a short linearly interpolated trace of the user’s viewpoint position in front of the rectangular screen. The analysis of the absolute position of the user’s head, shown in Figure A.7, indicates that the user’s horizontal position is typically in the middle section of the screen, covering around 70%. The vertical position corresponds to the height distribution of the subjects and leads to the assumption that the users do hardly move in this direction. The distance to the screen (z-dimension) is symmetrically distributed around the mean distance of 600 mm.

In addition to the absolute positioning of the users in relation to the screen, their movements are analyzed, which describe the change of position within a certain time frame. Since, the average interaction rate during the study is 1.13 Hz, the user’s movement is evaluated at 1 and 0.2 Hz. The data is shown in Figure
A.3 User viewpoint dynamics identification

A.9 and A.8. It is based on the same unit as the first one. In this case, zero represents ‘no movement’ within the time interval.

As already assumed from the absolute position measurements, the subjects move very little in the vertical dimension (see Figure A.9(b)) within 1 second, but they move along the horizontal axis (Figure A.9(a)), although the peak clearly indicates the inertia of the motion. Compared to the movement within a 5 seconds time frame (shown in Figure A.8), the user moves less in each direction, which is intuitively clear. This trend of less movement within shorter time frames continues up to the full sampling rate of 10 Hz.
A.4 Interaction precision

Studying the interaction precision of the user interacting with his bare finger and a pen TUI is described in the following section. Additional results of the study are reported by Droll [76].

A.4.1 Experiment design

The experiment is conducted on a large touch screen. The subjects must take position in front of the screen and click sequentially the illuminated targets which were arranged in a circular way (Fitts’ law test). The given task had to be done twice: Once interacting with the bare finger, and a second time with a pen TUI interaction device (tooltip < 2 mm in diameter). The click position and the target position on the screen is measured by the touch surface and saved in 2D display coordinates.

A.4.2 Setup

The interactive system used is a 50-inch plasma display from Pioneer (PDP-502 MXE) with a resistive touch-sensitive overlay from SMART technologies. Its resolution is 1024 x 768. This results in a pixel pitch of 1.072 in horizontal and 0.808 mm in vertical direction. The system consists of two layers. A display layer protected by a 3-5 mm glass pane and a touch-sensitive overlay with a distance of around 3 mm to the protective glass pane. The overall distance between the display plane and the touch plane is 10 mm. The origin of the display coordinate system is set to the upper left corner. The system’s x- and y-axis represent the plane coordinates in horizontal and vertical direction.

A.4.3 Subjects

The study is conducted with 17 students of the ETH Zurich. Out of the 17 participants, 16 were male and one was female. Their age ranges from 20 up to 32. Most of them do not wear glasses while five have glasses and three wear lenses. Their height is between 1670 mm up to 1930 mm and nearly all of them are right-handed. One subject is left-handed.

A.4.4 Task

First, the subjects make themselves familiar with the interactive system in order to achieve a feeling for the system’s touch behavior. Thereafter, they are told to stand in front of the screen. The subjects are free to use one or both hands to click the appearing targets on the interactive screen. The targets are 13 squares of 3 x 3 px arranged in a large circle of 600 mm in diameter, according to the Fitts’ Law tapping test [85]. The subjects must hit the single red square which changes with every interaction. The users are unaware of actually hitting or missing the target, as the red target changes as soon as a touch interaction is detected. When a click is detected, the click position and the target position are recorded. Every user must interact approximately 26 times, which represents hitting each target twice. This task is repeated twice: Once with their fingers and once with a pen, in order to detect differences between the interacting devices and the click accuracy.
A.4.5 Results

The results are presented in Section 5.5.4.
A Human computer interaction studies

Figure A.10: GUI button

(a) Macintosh  (b) Windows XP  (c) Plastique

Figure A.11: GUI checkboxes (Qt)

A.5 Focal point of area GUI target

The interaction position for different widget classes on different hardware setups with a minimal effect of parallax error is investigated, as discussed in detail by Reinhart [173]. In the following, the user’s behavior for representative common widget classes is analyzed: push buttons, checkboxes and hypermedia text references (links).

Push buttons are GUI elements for click interaction with a highlighted interaction area. Usually, rectangular buttons (see Figure A.10) contain a short text that describes the action which is executed on clicking the button. In contrast, the common function of checkboxes is to select an option. By clicking a checkbox, the user expects to (de)select the option which is indicated by a hook and described by a text as shown in Figure A.11. Hypermedia text references are text phrases that link to Hypermedia objects (i.e. web pages). Commonly, references are indicated by underlined and color-contrasting text. The text content defines the size of the interactive area and the border of links is not explicitly visible. Hence, the Geometric center is equal to the Optical center.

It is assumed that the interaction error is symmetric Normal distributed around the optical center of the widget. It is, moreover, assumed that the interaction error does not increase with the size of the widget.

A.5.1 Experiment design

The subjects are asked to fill out a questionnaire about their experience and interaction motivation before they interact on the hardware setups in random order.

Hardware setup

The interaction tests are randomly executed on three different hardware setups: First, on a large interactive LC-screen with a 65” image diagonal, a display resolution of 1920 x 1080 px and a pixel pitch of 1.33 \(\frac{\text{px}}{\text{mm}}\) (see Figure A.12(a)). The system is equipped with an optical tracking system (SMART\textsuperscript{®} Tech.\textsuperscript{TM}[20]) to detect the position of passive TUIs (pens) and the bare finger of the user to control the pointing device. The effect of the parallax error is minimized due to an offset between interaction and image plane of approximately 7 mm. Second, a 15” small interactive screen (Wacom\textsuperscript{TM}[27]) with inductive pen TUI tracking. The display resolution is 1024 x 768 px with a pixel pitch of 3.3 \(\frac{\text{px}}{\text{mm}}\). As shown in Figure A.12(b), the screen is located on a table with a height of 1000 mm. The third setup is a 15” non-interactive screen and a separate mouse pointing device. The hardware setup is shown in Figure A.12.
A.5 Focal point of area GUI target

![Interactive system setup](image)

**Figure A.12: Interactive system setup**

**Task**

The subjects are asked to fill out a questionnaire and run several tasks on the interactive systems, as described in the following section. In the questionnaire, the subjects are asked to sketch the aimed interaction position on different button types (see Section A.5.1).

**Test application**

The software application is based on the daily interaction on applications like automated teller machines or mobile phones. It consists of three parts.

In the selection test, the user is asked to select items from a list and finally press the 'done' button at the bottom of the widget. It is shown in Figure A.13(a).

The keypad test is shown in Figure A.13(a). The subjects are asked to type in their telephone number, birthday and the actual date in three setups, which differ in the text alignment (center, upper left, lower right) of the symmetrical push buttons. The text is alternately aligned in the upper left, in the center and in the lower right section.

The hyper reference test is shown in Figure A.13(c). The window shows a text with hyper references (blue, underlined phrases). The user is asked to click all references and finally hit the small 'done' push button at the end of the text.

The random test show a sequence of single widgets at random positions. Once the user clicks the currently shown widget, the next one is shown. Each widget is shown 8 times. Among others, 7 rectangular push buttons with a height of 50 and a width of 150, 225, 300 px and the two Label icons are shown.

The interaction test is implemented in the programming language Python on a Linux desktop machine, which runs the X.org windowing system [29] and the Qt graphic library [16]. The interaction software does not support any visual feedback of the pointing device to prevent 'hovering'.

**Widget classes**

The following widgets classes are analyzed: Wide (rectangular) push buttons of 50 × 150, 255 and 300 px, symmetrical (quadratic) push buttons of 50 × 50, 75 × 75 and 100 × 100 px as well as links of 18 ×
A Human computer interaction studies

Figure A.13: Application GUIs

16, 39, 51, 66, 80, 110 and 200 px and checkboxes with varying width, due to the length of the description text.

Subjects

The 5 female and 19 male subject have an average height of 1778 mm (standard deviation 103 mm) and an average age of 27.9 years (standard deviation 5 years). From the 24 users, 20 are right- and 4 are left-handed. The users are research assistants and students at the research lab at ETH Zurich as well as external people.

Measures

In the questionnaire the user was asked to sketch the aimed interaction position on push button targets. As shown in Figure A.14, the results are either the geometric or the optical center of the target. 100% of the subjects aim for the geometrical center of the target for links, on checkboxes only 23%.

While interacting on the screen, the user’s interaction is measured in terms of target type, position and size and the interaction position in display coordinates. Approximately 20,000 interactions for 35 different widget variants are recorded. The results are transferred into probability distributions by counting the interactions for each target class.

The accumulated probability distributions of the measurement data is tested to be Normal distributed with the Anderson-Darling test. The mean value of the interaction error is interpreted as focal point. The variance of the interaction error shows the stability of the interaction quality.
A.5 Focal point of area GUI target

Figure A.14: Aimed interaction position (questionnaire)
B

Viewpoint tracker implementation

B.1 Viewpoint tracker

Listing B.1: Viewpoint tracker C++ header

/*
 * ViewpointTracker
 * Created on: Nov 29, 2011
 * Author: miggeb@ethz.ch
 * the viewpoint is defined as 3d point VIEWPOINT_HEADTOP_OFFSET measures
 * below the highest point of the identified user
 * 3D coordinates:
 * z = distance to camera
 * x = horizontal camera coordinates
 * y = vertical camera coordinates
 * based on
 * PrimeSense NITE 1.3 - Players Sample
 * (Copyright (C) 2010 PrimeSense Ltd.)
 */

#ifndef VIEWPOINTTRACKER_H_
#define VIEWPOINTTRACKER_H_

#include <string>
#include <pthread.h>
#include <XnCppWrapper.h>
#include <XnOpenNI.h>
#include "coordinate_transform.h"
extern "C"
{
    #include "kinect_aux.h"
}

// ----------------
// Defines
// ----------------

#define VIEWPOINT_HEADTOP_OFFSET 150 // 15 cm beneath top of head
#define MIN(X,Y) ((X) < (Y) ? (X) : (Y))
B Viewpoint tracker implementation

```c
#define CHECK_RC(rc, what)    \    if (rc != XN_STATUS_OK)    \    {    \      printf("%s failed: %s\n", what, xnGetStatusString(rc));    \      return rc;    \    }

#define CHECK_ERRORS(rc, errors, what)    \    if (rc == XN_STATUS_NO_NODE_PRESENT)    \    {    \      XnChar strError[1024];    \      errors.ToString(strError, 1024);    \      printf("%s\n", strError);    \      return (rc);    \    }

#define XML_CONFIG_PATH "./Config/Config.xml"

// ----------------
// Class ViewpointTracker
// tracks the viewpoint of users with MS Kinect
// Coordinate System (positive direction): X sideways, Y upward, Z distance
// ----------------
class ViewpointTracker
{
    private:

    static const XnUInt16 N_JOINTS = 24;
    static const XnUInt16 N_USERS = 15;

    int nUsers; // number of detected users
    XnVector3D viewpoint_repos[N_USERS];
    pthread_mutex_t repos_mutex;
    pthread_t updateThread;

    xn::Context g_Context;
    xn::ScriptNode g_ScriptNode;
    xn::DepthGenerator g_DepthGenerator;
    xn::UserGenerator g_UserGenerator;

    XnUserID g_nPlayer;
    XnBool g_bCalibrated;
    XnStatus nRetVal;
    XnBool bPause;

    double smoothingFactor; // alpha value for exponential smoothing

    ViewpointTracker(); // init
    ViewpointTracker(ViewpointTracker const&){}; // copy
    //SkeletonTracker& operator=(SkeletonTracker const&){}; // assignment
    static ViewpointTracker* m_pInstance;

    // kinect control
    int initKinect();
    int shutdownKinect();
    int refreshKinect();
```

B.1 Viewpoint tracker

```c++
// user tracker
void FindPlayer();
void LostPlayer();
XnBool AssignPlayer(XnUserID user);

// repository
void initRepository();
void updateRepository(const xn::DepthMetaData& depthMD,
                       const xn::SceneMetaData& sceneMD);

// thread update callback
static void* updateLoop(void* args); // thread loop

public:
  static ViewpointTracker* Instance();
  ~ViewpointTracker();

// kinect control
void setKinectTiltAngle(int angle);
// spherical coordinates around z,x,y axis (y is invalid, since we use a g-
// sensor)
void getKinectOrientation(double *roll_z, double *pitch_x, double *acc_y);

// repository handler
void wipeRepository();
void printRepository();

// return viewpoints in camera coordinate system
int getViewpoints(XnVector3D* viewpointDestination, int n_viewpoints);

// return the number of detected users
int getNUsers();

// set smoothing parameter [0,1]
int setSmoothingFactor(double alpha);
// get smoothing parameter [0,1]
double getSmoothingFactor();

// openNI callbacks
static void XN_CALLBACK_TYPE NewUser(xn::UserGenerator& generator, XnUserID user, void* pCookie);
static void XN_CALLBACK_TYPE LostUser(xn::UserGenerator& generator, XnUserID user, void* pCookie);

};

#endif /* SKELETONTRACKER_H_ */
```

Listing B.2: Viewpoint tracker C++

/*
 * ViewpointTracker
 * Created on: Nov 29, 2011
 */
B Viewpoint tracker implementation

* Author: miggeb@ethz.ch
* /

#include "ViewpointTracker.h"
#include <XnTypes.h>

using namespace xn;

// ----------------
// Class
// ----------------
ViewpointTracker* ViewpointTracker::m_pInstance = NULL;

ViewpointTracker::ViewpointTracker() {
    nRetVal = XN_STATUS_OK;
    bPause = FALSE;
    g_bCalibrated = FALSE;
    g_nPlayer = 0;
    nUsers = 0;

    pthread_mutex_init(&repos_mutex, NULL);
    this->setSmoothingFactor(1.0f);
    this->initRepository();
    this->initKinect();

    // start thread
    pthread_create(&updateThread, NULL, &ViewpointTracker::updateLoop,
                   (void *) NULL);
}

ViewpointTracker::~ViewpointTracker() {
    bPause = TRUE;
    //pthread_join(updateThread);
    shutdownKinect();
    pthread_mutex_destroy(&repos_mutex);
}

// singleton
ViewpointTracker* ViewpointTracker::Instance() {
    if (!m_pInstance) // Only allow one instance of class to be generated.
        m_pInstance = new ViewpointTracker;

    return m_pInstance;
}

// connect kinect, register callbacks
int ViewpointTracker::initKinect() {
    XnStatus rc = XN_STATUS_OK;
    xn::EnumerationErrors errors;

    // init kinect sensor
    rc = g_Context.InitFromXmlFile(XML_CONFIG_PATH, g_ScriptNode, &errors);
    CHECK_ERRORS(rc, errors, "InitFromXmlFile");
    CHECK_RC(rc, "InitFromXml");

    rc = g_Context.FindExistingNode(XN_NODE_TYPE_DEPTH, g_DepthGenerator);
B.1 Viewpoint tracker

CHECK_RC(rc, "Find depth generator");
rc = g_Context.FindExistingNode(XN_NODE_TYPE_USER, g_UserGenerator);
CHECK_RC(rc, "Find user generator");
rc = g_Context.StartGeneratingAll();
CHECK_RC(rc, "StartGenerating");

// register callbacks
XnCallbackHandle hUserCBs;
g_UserGenerator.RegisterUserCallbacks(NewUser, LostUser, NULL, hUserCBs);

// init kinectAux
BM_KinectAux_openDevice(0);
BM_KinectAux_setLedOption(LED_OFF);
BM_KinectAux_setKinectTiltAngle(0);

return 0;

int ViewpointTracker::shutdownKinect() {
  g_ScriptNode.Release();
g_DepthGenerator.Release();
g_UserGenerator.Release();
g_Context.Release();

  return 0;
}

int ViewpointTracker::refreshKinect() {
  xn::SceneMetaData sceneMD;
  xn::DepthMetaData depthMD;

  // Read next available data
  nRetVal = g_Context.WaitAndUpdateAll();
  if (nRetVal != XN_STATUS_OK)
  {
    printf("UpdateData failed: %s\n", xnGetStatusString(nRetVal));
    return -1;
  }

  // Process the data
  g_DepthGenerator.GetMetaData(depthMD);
g_UserGenerator.GetUserPixels(0, sceneMD);
updateRepository(depthMD, sceneMD);

return 0;
}

// kinect control
void ViewpointTracker::setKinectTiltAngle(int angle) {
  BM_KinectAux_setKinectTiltAngle(angle);
}

void ViewpointTracker::getKinectOrientation(double *roll_z, double *pitch_x,
                                           double *acc_y) {
  BM_KinectAux_getOrientation(roll_z, pitch_x, acc_y);
}
### B Viewpoint tracker implementation

```cpp
void ViewpointTracker::initRepository()
{
    for(int userIndex = 0; userIndex < N_USERS; ++userIndex)
    {
        viewpoint_repos[userIndex].X = 0.0;
        viewpoint_repos[userIndex].Y = 0.0;
        viewpoint_repos[userIndex].Z = 0.0;
    }
}

// extract the head position from user tracker and depth camera
// VP = head top - VIEWPOINT_HEADTOP_OFFSET

void ViewpointTracker::updateRepository(const xn::DepthMetaData& depthMD,
                                        const xn::SceneMetaData& sceneMD)
{
    // Read in the MetaData
    const XnLabel* pLabelsRow = sceneMD.Data();
    const XnDepthPixel* pDepthRow = depthMD.Data();

    bool bUserDetected[N_USERS];
    int nUsersDetected = 0;
    XnPoint3D projectionPoint, worldPoint; // transformation helper

    XnVector3D gravity;
    double gravityLength;

    // setup normalized gravity vector
    BM_KinectAux_getGVector(&gravity);
    gravityLength = sqrt(pow(gravity.X, 2) + pow(gravity.Y, 2) + pow(gravity.Z, 2));
    gravity.X = gravity.X / gravityLength;
    gravity.Y = gravity.Y / gravityLength;
    gravity.Z = gravity.Z / gravityLength;

    for(int userIndex = 0; userIndex < N_USERS; ++userIndex)
    {
        bUserDetected[userIndex] = FALSE;
    }

    // kinect orientation
    XnVector3D viewpointTranslation;

    // clear and update repository
    pthread_mutex_lock(&repos_mutex);

    float alpha = getSmoothingFactor();

    // get head position from userID map and depth map
    for (XnUInt y = 0; y < depthMD.YRes(); ++y) {
        const XnLabel* pLabels = pLabelsRow;
        const XnDepthPixel* pDepth = pDepthRow;

        for (XnUInt x = 0; x < depthMD.XRes(); ++x, ++pLabels, ++pDepth) {
            if (*pLabels != 0 && bUserDetected[*pLabels] == FALSE) // user found and not already detected
            {
                // update repository

                // head top position = viewpointTranslation
                int viewPointIndex = (*pLabels) - 1;
            }
        }
    }
}
```
B.1 Viewpoint tracker

// convert data to real world coordinates: depthmap-> p1->p2
projectionPoint.X = x; projectionPoint.Y = y; projectionPoint.Z = *pDepth;
g_DepthGenerator.ConvertProjectiveToRealWorld(1,&projectionPoint,&worldPoint);

// translate viewpoint about VIEWPOINT_HEADTOP_OFFSET in gravity direction
// with exponential smooth with weight alpha
viewpoint_repos[viewPointIndex].X = (1.0 - alpha) * viewpoint_repos[viewPointIndex].X + alpha * (worldPoint.X - gravity.X * VIEWPOINT_HEADTOP_OFFSET);
viewpoint_repos[viewPointIndex].Y = (1.0 - alpha) * viewpoint_repos[viewPointIndex].Y + alpha * (worldPoint.Y - gravity.Y * VIEWPOINT_HEADTOP_OFFSET);
viewpoint_repos[viewPointIndex].Z = (1.0 - alpha) * viewpoint_repos[viewPointIndex].Z + alpha * (worldPoint.Z - gravity.Z * VIEWPOINT_HEADTOP_OFFSET);

// debug stuff
//printf("VP: %f\n",viewpoint_repos[viewPointIndex].Y);
//printf("HEAD: %f\n",worldPoint.Y);
bUserDetected[*pLabels] = TRUE;++nUsersDetected;
}
pLabelsRow += depthMD.XRes();
pDepthRow += depthMD.XRes();
}

pthread_mutex_unlock(&repos_mutex);

// visual at kinect
if (nUsersDetected > 0) {
BM_KinectAux_setLedOption(LED_BLINK_GREEN);
} else {
BM_KinectAux_setLedOption(LED_RED);
}

// CALLBACKS
// ---------------------------
XnBool ViewpointTracker::AssignPlayer(XnUserID user) {
if (g_nPlayer != 0) return FALSE;

XnPoint3D com;
g_UserGenerator.GetCoM(user, com);
if (com.Z == 0) return FALSE;

printf("Matching for existing calibration\n");
g_UserGenerator.GetSkeletonCap().LoadCalibrationData(user, 0);
g_UserGenerator.GetSkeletonCap().StartTracking(user);
B Viewpoint tracker implementation

```cpp
// Viewpoint tracker implementation

g_nPlayer = user;
return TRUE;
}

int ViewpointTracker::setSmoothingFactor(double alpha) {
  if (alpha < 0.0 || alpha > 1.0) {
    printf("value out of bounds: \d", alpha);
    return -1;
  }
  this->smoothingFactor = alpha;
  return 0;
}

double ViewpointTracker::getSmoothingFactor() {
  return this->smoothingFactor;
}

//////// CALLBACKS //////////

void XN_CALLBACK_TYPE ViewpointTracker::NewUser(xn::UserGenerator& generator, 
                                      XnUserID user, void* pCookie) {
  printf("New user \d\n", user);
  (ViewpointTracker::m_pInstance->nUsers)++;
  //ViewpointTracker::nUsers++;
}

void XN_CALLBACK_TYPE ViewpointTracker::LostUser(xn::UserGenerator& generator, 
                                       XnUserID user, void* pCookie) {
  printf("Lost user \d\n", user);
  (ViewpointTracker::m_pInstance->nUsers)--;
}

void ViewpointTracker::wipeRepository() {
  pthread_mutex_lock(&repos_mutex);
  // wipe data structures
  for(int userIndex = 0; userIndex < N_USERS; ++userIndex) {
    viewpoint_repos[userIndex].X = NAN;
    viewpoint_repos[userIndex].Y = NAN;
    viewpoint_repos[userIndex].Z = NAN;
  }
  pthread_mutex_unlock(&repos_mutex);
}

void ViewpointTracker::printRepository() {
  int userIndex;
  for(userIndex = 0; userIndex < N_USERS; ++userIndex) {
    printf("[\f,\f,\f \]", viewpoint_repos[userIndex].X, viewpoint_repos[userIndex ].
    Y, viewpoint_repos[userIndex].Z);
  }
}
B.1 Viewpoint tracker

// copy repository
int ViewpointTracker::getViewpoints(XnVector3D* viewpointDestination,
    int n_viewpoints) {
    int n_copiedViewpoints, length;

    pthread_mutex_lock(&repos_mutex);
    n_copiedViewpoints = MIN(n_viewpoints, N_USERS);
    length = n_copiedViewpoints * sizeof(XnVector3D);
    memcpy(viewpointDestination, &viewpoint_repos, length);
    pthread_mutex_unlock(&repos_mutex);

    return n_copiedViewpoints;
}

int ViewpointTracker::getNUsers() {
    return nUsers;
}

void* ViewpointTracker::updateLoop(void* args) {
    while (!m_pInstance->bPause)
    {
        m_pInstance->refreshKinect();
    }
}

B.1.1 Helper functions

Listing B.3: Coordinate transformation header

/*@ coordinate_transform.h
* Copyright (C) 2011 Christoph Bubenhofer
* This file is part of SkeletonTracker.
* SkeletonTracker is free software: you can redistribute it and/or modify
* it under the terms of the GNU Lesser General Public License as published
* by the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
* SkeletonTracker is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU Lesser General Public License for more details.
* You should have received a copy of the GNU Lesser General Public License
* along with OpenNI. If not, see <http://www.gnu.org/licenses/>.
*---------------------------------------------------------------------*/

#ifndef COORDINATE_TRANSFORM_H_
#define COORDINATE_TRANSFORM_H_

#include <math.h>
#include <XnTypes.h>
extern "C"
{
    #include "kinect_aux.h"
}
B Viewpoint tracker implementation

// rotate to earth coordinate system using the Kinect accelerometer
void ST_setEarthOrientation(XnVector3D *position);

// to transform the coordinate system
XnVector3D ST_coordinateTransform(XnVector3D position);

// translations
void ST_setOrigin(XnVector3D *position, XnVector3D origin);
void ST_setRotation(XnVector3D *position, double alpha, double beta, double delta);

// scaling
void ST_setScale(XnVector3D *position, double scalingFactor);

Listing B.4: Coordinate transformation

#include "coordinate_transform.h"

int BM_SMOOTH_ACC = 10; // for smoothing the accelerometer data

void __ST_multiplicationMatrix3x3Matrix3x3(double A[], double B[], double C[]);
void __ST_multiplicationMatrix3x3Vector3x1(double A[], double b[], double c[]);

void __ST_multiplicationMatrix3x3Matrix3x3(double A[], double B[], double C[])
{
    for (int row = 0; row < 3; row++) {
        for (int column = 0; column < 3; column++) {
        }
    }
}
\[ \text{fact that it is a 3x3 matrix} \]

// for 3x3 matrix saved in an array double[9] and a vector with 3 rows
void __ST_multiplicationMatrix3x3Vector3x1(double A[], double b[], double c[])
{
    for (int row = 0; row < 3; ++row)
    {
    }
}

XnVector3D ST_coordinateTransform(XnVector3D position) {
    double distanceKinTo0X = 0; // distance to from Kinect to 0 in mm
    double distanceKinTo0Y = 0;
    double distanceKinTo0Z = 0;
    double alpha, beta, gamma;
    XnVector3D g;
    int horizontalAngle = 0;
    double delta = (double) horizontalAngle / 180 * M_PI;
    double neigung = 0;
    // printf("\nDie Winkel alpha und alpha: %f\n",delta/M_PI*180);
    double cosAlpha, sinAlpha, cosBeta, sinBeta, cosDelta, sinDelta,
        cosNeigung, sinNeigung;
    // delta: angles to turn around y system of earth in rad
    BM_KinectAux_getOrientation(&alpha,&beta,&gamma);
    cosAlpha = cos(alpha);
    sinAlpha = sin(alpha);
    cosBeta = cos(beta);
    sinBeta = sin(beta);
    cosDelta = cos(delta);
    sinDelta = sin(delta);
    cosNeigung = cos(neigung);
    sinNeigung = sin(neigung);
    // rotation matrixes around x,y,z
    double rotationX[9] = { 1, 0, 0,
        0, cosBeta, sinBeta,
        0, -sinBeta, cosBeta };}

double rotationZ[9] = { cosAlpha, sinAlpha, 0,
        -sinAlpha, cosAlpha, 0,
        0, 0, 1 };}
double rotationY[9] = { cosDelta, 0, -sinDelta,
        0, 1, 0,
        sinDelta, 0, cosDelta };}
    // calculate rotation matrix for three dimensions in rotating order Z,X,Y

185
**B Viewpoint tracker implementation**

```c
double rotationXZ[9];
double rotationYXZ[9];
__ST_multiplicationMatrix3x3Matrix3x3(rotationX, rotationZ, rotationXZ);
__ST_multiplicationMatrix3x3Matrix3x3(rotationY, rotationXZ, rotationYXZ);

double positionVector[3];
positionVector[0] = position.X;

double correctedPositionVector[3];  // multiplying Vector with TransformMatrix
__ST_multiplicationMatrix3x3Vector3x1(rotationYXZ, positionVector, correctedPositionVector);
correctedPositionVector[0] = correctedPositionVector[0] - distanceKinTo0X;  // correcting with the distance from Kinect to 0

// rotation matrixes around x,y,z
double rotationN[9] = { 1, 0, 0,
                      0, cosNeigung, sinNeigung,
                      0, -sinNeigung, cosNeigung };

double correctedPositionVectorNeigung[3];
__ST_multiplicationMatrix3x3Vector3x1(rotationN, correctedPositionVector, correctedPositionVectorNeigung);
XnVector3D correctedPosition;  // saving into return value
correctedPosition.X = correctedPositionVectorNeigung[0];
correctedPosition.Y = correctedPositionVectorNeigung[1];
correctedPosition.Z = correctedPositionVectorNeigung[2];
return correctedPosition;
```

```c
// translate vector
void ST_setOrigin(XnVector3D *position, XnVector3D origin)
{
    // newPosition = position - origin
    position->X = position->X - origin.X;
    position->Y = position->Y - origin.Y;
    position->Z = position->Z - origin.Z;
}

void ST_setEarthOrientation(XnVector3D *position)
{
    throw "not implemented!";
}

// rotate around delta (y), alpha (z), beta (x) [RAD]
void ST_setRotation(XnVector3D *position, double alpha, double beta, double delta)
{
    double positionVector[3] = (position->X, position->Y, position->Z);
    double correctedPositionVector[3];
    double cosAlpha, sinAlpha, cosBeta, sinBeta, cosDelta, sinDelta;
```
cosAlpha = cos(alpha);
sinAlpha = sin(alpha);

cosBeta = cos(beta);
sinBeta = sin(beta);

cosDelta = cos(delta);
sinDelta = sin(delta);

// rotation matrixes around x, y, z
double rotationX[9] = { 1, 0, 0,
                       0, cosBeta, sinBeta,
                       0, -sinBeta, cosBeta };

double rotationZ[9] = { cosAlpha, sinAlpha, 0,
                       -sinAlpha, cosAlpha, 0,
                       0, 0, 1 };

double rotationY[9] = { cosDelta, 0, -sinDelta,
                       0, 1, 0,
                       sinDelta, 0, cosDelta };

// calculate rotation matrix for three dimensions in rotating order Z, X, Y
double rotationXZ[9];
double rotationYXZ[9];
__ST_multiplicationMatrix3x3Matrix3x3(rotationX, rotationZ, rotationXZ);
__ST_multiplicationMatrix3x3Matrix3x3(rotationY, rotationXZ, rotationYXZ);
__ST_multiplicationMatrix3x3Vector3x1(rotationYXZ, positionVector,
                                      (double*)&correctedPositionVector);

position->X = correctedPositionVector[0];
position->Y = correctedPositionVector[1];
position->Z = correctedPositionVector[2];
}

// scale vector
void ST_setSkale(XnVector3D *position, double scalingFactor)
{
    position->X *= scalingFactor;
    position->Y *= scalingFactor;
    position->Z *= scalingFactor;
}

Listing B.5: AUX header

/*
 * kinect_aux.h
 *
 * Created on: May 13, 2011
 * Author: Bastian migge <miggeb@ethz.ch>
 *
 * Based on kinect_aux http://www.ros.org/wiki/kinect
 */
#ifndef _KINECT_AUX_
#define _KINECT_AUX_

#include <stdio.h>
#include <math.h>

#endif

#include <stdio.h>
#include <math.h>
B Viewpoint tracker implementation

```c
#include <unistd.h>
#include <libusb-1.0/libusb.h>
#include <XnTypes.h>
#define BM_PI 3.141582

// VID and PID for Kinect and motor/acc/leds
#define MS_MAGIC_VENDOR 0x45e
#define MS_MAGIC_MOTOR_PRODUCT 0x02b0
// Constants for accelerometers
#define GRAVITY 9.80665
#define FREENECT_COUNTS_PER_G 819.
// The kinect can tilt from +31 to -31 degrees in what looks like 1 degree
// increments
// The control input looks like 2*desired_degrees
#define MAX_TILT_ANGLE 31.
#define MIN_TILT_ANGLE (-31.)

/// Enumeration of LED states
/// See http://openkinect.org/wiki/Protocol_Documentation#Setting_LED for more
/// information.
typedef enum
{
    LED_OFF = 0, /**< Turn LED off */
    LED_GREEN = 1, /**< Turn LED to Green */
    LED_RED = 2, /**< Turn LED to Red */
    LED_YELLOW = 3, /**< Turn LED to Yellow */
    LED_BLINK_GREEN = 4, /**< Make LED blink Green */
    LED_BLINK_RED_YELLOW = 6, /**< Make LED blink Red/Yellow */
} freenect_led_options;

/// Enumeration of tilt motor status
typedef enum
{
    TILT_STATUS_STOPPED = 0x00, /**< Tilt motor is stopped */
    TILT_STATUS_LIMIT = 0x01, /**< Tilt motor has reached movement limit */
    TILT_STATUS_MOVING = 0x04, /**< Tilt motor is currently moving to new position */
} freenect_tilt_status_code;

struct BM_KinectAuxState {
    int accelerometer_x;
    int accelerometer_y;
    int accelerometer_z;
    int tilt_angle;
    int tilt_status;
};

int BM_KinectAux_init(int deviceIndex);
void BM_KinectAux_close();

int BM_KinectAux_openDevice(int index);
void BM_KinectAux_getState(struct BM_KinectAuxState* state);
void BM_KinectAux_setKinectTiltAngle(const double angleTarget);
void BM_KinectAux_setLedOption(const freenect_led_options option);
```
// get G vector of length 1
void BM_KinectAux_getGVector(XnVector3D *g);

// get device orientation in spherical coordinates [RAD]
void BM_KinectAux_getOrientation(double *roll_z, double *pitch_x, double *acc_y);

// Christoph’s implementation
void CB_KinectAux_getOrientation(double *alpha, double *beta, double *gamma);

/**
 * Get the axis-based gravity adjusted accelerometer state, as laid
 * out via the accelerometer data sheet, which is available at
 * @param state State to extract accelerometer data from
 * @param x Stores X-axis accelerometer state
 * @param y Stores Y-axis accelerometer state
 * @param z Stores Z-axis accelerometer state
 */
void freenect_get_mks_accel(struct BM_KinectAuxState *state, double * x, double * y, double * z);

// shake your boody :-)
void BM_KinectAux_shake(int target);

#endif /* _KINECT_AUX_ */

Listing B.6: AUX

/*
 * kinect_aux.c
 * Created on: May 13, 2011
 * Author: Bastian migge <miggeb@ethz.ch>, Christoph Bubenhofer
 * Based on kinect_aux http://www.ros.org/wiki/kinect
 */

#include "kinect_aux.h"

// --- DEFINES ---
#define BM_SMOOTH_ACC 10 //for smoothing the accelerometer data

// --GLOBALS--
libusb_device_handle *dev;

// ---CODE---
int BM_KinectAux_openDevice(int index)
{
    libusb_device **devs; //pointer to pointer of device, used to retrieve a list of devices
    ssize_t cnt = libusb_get_device_list (0, &devs); //get the list of devices
    if (cnt < 0)
    {
        printf("No device on USB");
        return -1;
    }

    int nr_mot = 0;
    int i;
for (i = 0; i < cnt; ++i) {
    struct libusb_device_descriptor desc;
    const int r = libusb_get_device_descriptor(devs[i], &desc);
    if (r < 0)
        continue;

    // Search for the aux
    if (desc.idVendor == MS_MAGIC_VENDOR && desc.idProduct == MS_MAGIC_MOTOR_PRODUCT) {
        // If the index given by the user matches our camera index
        if (nr_mot == index) {
            if ((libusb_open(devs[i], &dev) != 0) || (dev == 0)) {
                printf("Cannot open aux %i", index);
                return;
            }
            // Claim the aux
            libusb_claim_interface(dev, 0);
            break;
        } else {
            nr_mot++;
        }
    } else {
        nr_mot++;
    }
}
libusb_free_device_list(devs, 1); // free the list, unref the devices in it
return 0;

void ST_KinectAux_getState(struct BM_KinectAuxState* state) {
    uint8_t buf[10];
    const int ret = libusb_control_transfer(dev, 0xC0, 0x32, 0x0, 0x0, buf, 10, 0);
    if (ret != 10) {
        printf("Error in accelerometer reading, libusb_control_transfer returned %i", ret);
        return;
    }

    // extract data
    const int16_t ux = ((int16_t)buf[2] << 8) | buf[3];
    const int16_t uy = ((int16_t)buf[4] << 8) | buf[5];
    const int16_t uz = ((int16_t)buf[6] << 8) | buf[7];
    const int16_t accelerometer_x = (int16_t)ux;
    const int16_t accelerometer_y = (int16_t)uy;
    const int16_t accelerometer_z = (int16_t)uz;
    const int8_t tilt_angle = (int8_t)buf[8];
    const int tilt_status = buf[9];

    // pack data
    state->accelerometer_x = accelerometer_x;
    state->accelerometer_y = accelerometer_y;
    state->accelerometer_z = accelerometer_z;
    state->tilt_angle = tilt_angle;
B.1 Viewpoint tracker

```c
state->tilt_status = tilt_status;
}

void BM_KinectAux_setKinectTiltAngle(const double angleTarget)
{
    uint8_t empty[0x1];
    double angle;
    angle = (angleTarget < MIN_TILT_ANGLE) ? MIN_TILT_ANGLE : ((angleTarget > MAX_TILT_ANGLE) ? MAX_TILT_ANGLE : angleTarget);
    angle = angle * 2;
    const int ret = libusb_control_transfer(dev, 0x40, 0x31, (uint16_t)angle, 0x0, empty, 0x0, 0);
    if (ret != 0)
    {
        printf("Error in setting tilt angle, libusb_control_transfer returned %i", ret);
        return;
    }
}

void BM_KinectAux_setLedOption(const freenect_led_options option)
{
    uint8_t empty[0x1];
    const int ret = libusb_control_transfer(dev, 0x40, 0x06, (uint16_t)option, 0x0, empty, 0x0, 0);
    if (ret != 0)
    {
        printf("Error in setting LED options, libusb_control_transfer returned %i", ret);
        return;
    }
}

// source from https://github.com/OpenKinect/libfreenect.git/fakenect/fakenect.c
void freenect_get_mks_accel(struct BM_KinectAuxState *state, double* x, double* y, double* z) {
    //the documentation for the accelerometer (http://www.kionix.com/ProductSheets/KXSD9%20Product%20Brief.pdf)
    //states there are 819 counts/g
    *x = (double) state->accelerometer_x / FREENECT_COUNTS_PER_G * GRAVITY;
    *y = (double) state->accelerometer_y / FREENECT_COUNTS_PER_G * GRAVITY;
    *z = (double) state->accelerometer_z / FREENECT_COUNTS_PER_G * GRAVITY;
}

int BM_KinectAux_init(int deviceIndex)
{
    int ret = libusb_init(0);
    if (ret)
    {
        return 1;
    }

    BM_KinectAux_openDevice(deviceIndex);

    if (!dev)
    {
```

B Viewpoint tracker implementation

```c
printf("No valid aux device found");
libusb_exit(0);
return 2;
}
return 0;
}
void BM_KinectAux_close()
{
libusb_exit(0);
}

// get orientation in [RAD]
void BM_KinectAux_getGVector(XnVector3D *g)
{
    struct BM_KinectAuxState kstate;
    ST_KinectAux_getState(&kstate);
    double gX, gY, gZ;
    double length;
    // g-vector in kinect coordinates system
    gX = ((double)(kstate.accelerometer_x)/FREENECT_COUNTS_PER_G);
    gY = ((double)(kstate.accelerometer_y)/FREENECT_COUNTS_PER_G);
    gZ = ((double)(kstate.accelerometer_z)/FREENECT_COUNTS_PER_G);
    length = sqrt(gX*gX+gY*gY+gZ*gZ);
    gX /= length;
    gY /= length;
    gZ /= length;
    g->X = gX;
    g->Y = gY;
    g->Z = gZ;
}

// source is based on http://abstrakraft.org/cwiid/browser/wmgui/main.c
void BM_KinectAux_getOrientation(double *roll_z, double *pitch_x, double *acc_y)
{
    XnVector3D g;
    BM_KinectAux_getGVector(&g);
    *acc_y = sqrt(pow(g.X,2)+pow(g.Z,2)+pow(g.Y,2));
    *roll_z = atan(g.X/g.Y);
    if (g.Y <= 0.0)
    {
        *roll_z += BM_PI * ((g.X > 0.0) ? 1 : -1);
    }
    *roll_z += -1;
    *pitch_x = atan(g.Z/g.Y*cos(*roll_z));
}

// Christoph's orientation calculation
void CB_KinectAux_getOrientation(double *alpha, double *beta, double *gamma)
{
    const int X_VALUE_MAX= 815; // max measured values for norming
    const int Y_VALUE_MAX= 759;
    const int Z_VALUE_MAX= 815;
    struct BM_KinectAuxState oneState;
    double accelerometerDataX=0;
```

192
double accelerometerDataY=0;
double accelerometerDataZ=0;
int statecount;

for (statecount=0; statecount < BM_SMOOTH_ACC; statecount++)
{
    ST_KinectAux_getState(&oneState);
    accelerometerDataX+=oneState.accelerometer_x;
    accelerometerDataY+=oneState.accelerometer_y;
    accelerometerDataZ+=oneState.accelerometer_z;
}
accelerometerDataX=accelerometerDataX/(BM_SMOOTH_ACC*X_VALUE_MAX); // divided through the Smoothing factor and the max_value for norming
accelerometerDataY=accelerometerDataY/(BM_SMOOTH_ACC*Y_VALUE_MAX);
accelerometerDataZ=accelerometerDataZ/(BM_SMOOTH_ACC*Z_VALUE_MAX);

*gamma=-atan(accelerometerDataX/sqrt(pow(accelerometerDataZ,2)+pow(accelerometerDataY,2))); // calculation of the right angles for correction to horizontal plane
*beta=atan(accelerometerDataZ/sqrt(pow(accelerometerDataX,2)+pow(accelerometerDataY,2)));
*alpha=asin(sin(*gamma)/cos(*beta));

void BM_KinectAux_shake(int target)
{
    BM_KinectAux_setKinectTiltAngle(MAX_TILT_ANGLE);
sleep(3);
    BM_KinectAux_setKinectTiltAngle(MIN_TILT_ANGLE);
sleep(3);
    BM_KinectAux_setKinectTiltAngle(target);
}

// example update loop - could be used for pthread.
int updateLoop(int argc, char* argv[])
{
    struct BM_KinectAuxState state;
    BM_KinectAux_init(0);

    while (1)
    {
        sleep(1);
        ST_KinectAux_getState(&state);
        printf("%d,%d,%d,%d,%d\n",state.accelerometer_x,state.accelerometer_y, state.accelerometer_z,state.tilt_angle,state.tilt_status);
    }
    BM_KinectAux_close();
    return 0;
}

B.1.2 Python interface

Listing B.7: Viewpoint tracker C

#include "ViewpointTracker.h"

// C function for ctypes python interface
extern "C"
{
    ViewpointTracker* ViewpointTracker_instance(){
return ViewpointTracker::Instance();
}

void ViewpointTracker_setSmoothing(ViewpointTracker* tracker, double alpha) {
    tracker->setSmoothingFactor(alpha);
}

void ViewpointTracker_getSmoothing(ViewpointTracker* tracker, double* alpha) {
    *alpha = tracker->getSmoothingFactor();
}

void ViewpointTracker_setKinectTiltAngle(ViewpointTracker* tracker, int angle) {
    tracker->setKinectTiltAngle(angle);
}

void ViewpointTracker_getKinectOrientation(ViewpointTracker* tracker, double* roll_z, double* pitch_x, double* acc_y) {
    tracker->getKinectOrientation(roll_z, pitch_x, acc_y);
}

void ViewpointTracker_wipeRepository(ViewpointTracker* tracker) {
    tracker->wipeRepository();
}

void ViewpointTracker_printRepository(ViewpointTracker* tracker) {
    tracker->printRepository();
}

int ViewpointTracker_getViewpoint(ViewpointTracker* tracker, int id, double* x, double* y, double* z) {
    if (id > 24)
        return -1;

    XnVector3D vps[24];
    tracker->getViewpoints((XnVector3D*) &vps, 24);
    *x = (double) vps[id].X;
    *y = (double) vps[id].Y;
    *z = (double) vps[id].Z;
    return 0;
}

int ViewpointTracker_getNumberOfUsers(ViewpointTracker* tracker)
{
    return tracker->getNUUsers();
}

Listing B.8: Python viewpoint tracker

# -*- coding: utf-8 -*-
***
pykinecttracker.kinect
~~~~~~
:copyright: (c) 2010,2011 by Bastian Migge
:license: BSD3, see LICENSE for more details.
:description: Python wrapper of C++ Viewpoint tracker
***

from ctypes import *
import time
from pykinecttracker.core import ViewpointTracker

class ViewpointTrackerKinect(ViewpointTracker):
    ''' Python wrapper class for viewpoint tracker '''

    def __init__(self,libraryPath,**kwargs):
        ''' initialize arguments
        libraryPath - path string of c library
        '''

        super(ViewpointTrackerKinect,self).__init__(**kwargs)

        ViewpointTrackerKinect._trackerLib = cdll.LoadLibrary(libraryPath)
        self.obj = ViewpointTrackerKinect._trackerLib.ViewpointTracker_instance()

    def setKinectTiltAngle(self,angle):
        ViewpointTrackerKinect._trackerLib.ViewpointTracker_setKinectTiltAngle(
            self.obj,angle)

    def getKinectOrientation(self):
        ''' return the kinect orientation
        return
        orientation - [x,y,z]
        '''

        x, y, z = c_double(),c_double(),c_double()
        ViewpointTrackerKinect._trackerLib.ViewpointTracker_getKinectOrientation(
            self.obj,
            byref(x),
            byref(y),
            byref(z))

        orientation = [x.value,y.value,z.value]
        return orientation

    def wipeRepository(self):
        ''' wipe the tracker repository '''
        ViewpointTrackerKinect._trackerLib.ViewpointTracker_wipeRepository(self.obj)

    def printRepository(self):
        ''' wipe the tracker repository '''
        ViewpointTrackerKinect._trackerLib.ViewpointTracker_printRepository(self.obj)

    def getViewpoint(self,id=0):
        ''' return the viewpoint with ID id
        arguments
        id - viewpoint id [int]
        return
        viewpoint - [x,y,z]
        '''

        x, y, z = c_double(),c_double(),c_double()
        status = ViewpointTrackerKinect._trackerLib.ViewpointTracker_getViewpoint(
            self.obj,
            id,
            byref(x),
            byref(y),
            byref(z))
B Viewpoint tracker implementation

```python
if (status < 0):
    raise Exception("Error gathering viewpoint")

viewpoint = [x.value, y.value, z.value]
return viewpoint

def getNumberOfUsers(self):
    ''' return the number of detected users '''
    return ViewpointTrackerKinect._trackerLib.ViewpointTracker_getNumberOfUsers(self.obj)

def getSmoothingFactor(self):
    ''' smoothing factor
    arguments:
    alpha - smoothing factor [0,1] (1 = no smoothing)
    '''
    alpha = c_double()
    ViewpointTrackerKinect._trackerLib.ViewpointTracker_getSmoothing(self.obj, byref(alpha))
    return alpha.value

def setSmoothingFactor(self, alpha):
    ''' set factor of exponential smoothing
    arguments:
    alpha - smoothing factor [0,1] (1 = no smoothing)
    '''
    return ViewpointTrackerKinect._trackerLib.ViewpointTracker_setSmoothing(self.obj, c_double(alpha))

def main():
    k = ViewpointTrackerKinect(libraryPath="lib/libSkeletonTrackerKinect.so")
    time.sleep(4)

    while (1):
        for id in range(k.getNumberOfUsers()):
            print(id,",", k.getViewpoint(id))
            time.sleep(0.1)

if __name__ == '__main__':
    main()
```
B.2 Parallax error correction controller prototype

B.2.1 Graphical user interfaces

Window managers (windowing systems) provide the basic programming model for showing digital content on the screen and processing the user’s input. To allow the user to switch between multiple contents on the limited display plane, they show multiple overlapping windows at the same time [118]. This idea was implemented collectively by Xerox Star, Apple Macintosh and Microsoft Windows [151] as *graphical toolkit* and is, today, common on desktop computers and laptops. A widely used implementation on UNIX machines is the X window manager (x.org). Since the window manager is closely coupled to the operating system, which provides hardware access to the user space applications and manages the application execution, the window manager is mostly seen as part of the operating system, like in Microsoft Windows. Due to the small display of mobile phones and tablets, applications are almost exclusively shown at full screen. Hence, most mobile operating systems, for instance Android or Apple iOS, do not implement a window manager, although the GUI widget toolkit provides several views for the purpose of mobile devices, i.e. small displays and touch screens.

On top of the window manager, the *widget toolkit* provides a library of reusable widgets which supersedes the effort of implementing the graphical elements from scratch for each application, reduces the effort of the user to learn new interface principles and, hence, supports the idea of standards for user interfaces\(^1\). Although this ended up at most in guidelines, common implementations are Cocoa on Mac OS X, Windows Forms and Windows Presentation Foundation on Microsoft Windows and GTK, Qt as cross platform implementation based on C,C++ as well as AWT, Swing and SWT for applications written in the programming language Java. Other graphical toolkits, like Quartz or OpenGL, focus on two-dimensional respectively three-dimensional image and video rendering.

B.2.2 Software prototype in PyQt

On today’s common computer systems with graphical user interfaces (GUI), the system architecture is set up as event chain from the hardware driver to the application logic. The windowing system provides user pointing interaction information to the active application by sending button or motion events. The application forwards the information in the widget framework to the most upper widget (and window)

---

\(^1\)IBM, Systems Application Architecture: Common User Access

**Figure B.1:** GUI structure
which is positioned under the interaction position. At the widget, the interaction event is observed and enters the application logic of the user application. If the widget does not accept the event, it is forwarded to the parent widget up to the root window. The process chain is shown in Figure B.2.

The parallax correction system is integrated into the event chain between pointing device driver and user specific application as shown in Figure B.2. The actual correction settings are applied to the pointing device coordinates and forwarded to the application within the system’s event chain. To correct the 2-dimensional pointing position, the correction controller requires getting and changing the interaction position and the target position to query the decision maker for the correction offset. As shown in Figure B.2, the application has access to the pointing event before it actually achieves the window. Hence, the widget and the application can query the widget framework of the GUI application, correcting the pointing position is implemented within the application and not at the operating or the windowing system.

The prototype is implemented on a Linux system, with the Window Manager X.Org [29] and the cross-platform Widget framework Qt [16]. The X.Org server provides (among others) the interaction events *XButtonEvent* and *XMotionEvent* as a structured data type to the application via the method called *x11EventFilter()*. At this point, the application queries the correction controller and changes the event position information transparently to the GUI component of the application. In Qt, the application can query its windows and widget directly (*QApplication.widgetAt()) to eventually calculate the resulting interaction error, which is forwarded to the correction decision maker. The event chain of correcting the pointing position is shown in Figure B.2.

The correction controller is implemented in two components. The correction component and the preprocessor are described in the following.

**Correction component**

The correction component intercepts the pointing event at the interface between X.Org server and software application. It applies the correction and forwards the event to the corresponding GUI window.

**Listing B.9: PyXevent header**

```c
/****************************
```
B.2 Parallax error correction controller prototype

X11 Event module bridge to Python

Copyright 2010 by Bastian Migge <miggeb@ethz.ch>

This X11 event bridges the x11 events as C structs to Python

A C extension module for Python, called "Xevent", to translate X11
events into python dictionaries.

License

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Links

http://nedbatchelder.com/text/whirlext.html
http://docs.python.org/release/1.5.2/api/
http://docs.python.org/release/2.5.2/api/cObjects.html

Listing B.10: PyXevent source

---

X11 Event module bridge to Python

Copyright 2010 by Bastian Migge <miggeb@ethz.ch>

This X11 event bridges the x11 events as C structs to Python

---
B Viewpoint tracker implementation

A C extension module for Python, called "Xevent", to translate X11 events into python dictionaries.

compile this into a shared library ".so" on python path, import Xevent;
(gcc Xevent.c -g -l$PYTHONINCLUDEPATH -lX11 -fpic -shared -o Xevent.so)

License

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Links

http://nedbatchelder.com/text/whirlext.html
http://docs.python.org/release/1.5.2/api/
http://docs.python.org/release/2.5.2/api/cObjects.html
*********************************************************************/

#include "PyXevent.h"

// get window information from X11
static PyObject* _examineWindow(XAnyEvent *xEvent);

static PyObject* getEventAsPyDict(PyObject *self, PyObject *args)
{
PyObject *pyCobj; /* python’s void pointer ref of the x11 event */
PyObject *pyDict; /* return information as a dict */
PyObject *pyWindowDict; /* window information */

XAnyEvent *xEvent = (XAnyEvent *)PyCObject_AsVoidPtr(pyCobj);

switch (xEvent->type) {
case ButtonPress:
    xButtonEvent = (XButtonEvent *)xEvent;
    // examine cursor
    PyDict_SetItemString(pyDict, "type", Py_BuildValue("s", "MousePressEvent"));
}
B.2 Parallax error correction controller prototype

```python
PyDict_SetItemString(pyDict, "button", Py_BuildValue("i", xButtonEvent->button));
PyDict_SetItemString(pyDict, "x", Py_BuildValue("i", xButtonEvent->x));
PyDict_SetItemString(pyDict, "y", Py_BuildValue("i", xButtonEvent->y));
PyDict_SetItemString(pyDict, "globalX", Py_BuildValue("i", xButtonEvent->x_root));
PyDict_SetItemString(pyDict, "globalY", Py_BuildValue("i", xButtonEvent->y_root));

// examine window
pyWindowDict = _examineWindow(xEvent);
PyDict_SetItemString(pyDict, "window", pyWindowDict);
break;

case ButtonRelease:
    xButtonEvent = (XButtonEvent *) xEvent;

    // examine cursor
    PyDict_SetItemString(pyDict, "type", Py_BuildValue("s", "MouseReleaseEvent"));
    PyDict_SetItemString(pyDict, "button", Py_BuildValue("i", xButtonEvent->button));
    PyDict_SetItemString(pyDict, "x", Py_BuildValue("i", xButtonEvent->x));
    PyDict_SetItemString(pyDict, "y", Py_BuildValue("i", xButtonEvent->y));
    PyDict_SetItemString(pyDict, "globalX", Py_BuildValue("i", xButtonEvent->x_root));
    PyDict_SetItemString(pyDict, "globalY", Py_BuildValue("i", xButtonEvent->y_root));

    // examine window
    pyWindowDict = _examineWindow(xEvent);
    PyDict_SetItemString(pyDict, "window", pyWindowDict);
    break;

case MotionNotify:
    xMotionEvent = (XMotionEvent *) xEvent;

    // examine cursor
    PyDict_SetItemString(pyDict, "type", Py_BuildValue("s", "MouseMoveEvent"));
    PyDict_SetItemString(pyDict, "x", Py_BuildValue("i", xMotionEvent->x));
    PyDict_SetItemString(pyDict, "y", Py_BuildValue("i", xMotionEvent->y));
    PyDict_SetItemString(pyDict, "globalX", Py_BuildValue("i", xMotionEvent->x_root));
    PyDict_SetItemString(pyDict, "globalY", Py_BuildValue("i", xMotionEvent->y_root));

    // examine window
    pyWindowDict = _examineWindow(xEvent);
    PyDict_SetItemString(pyDict, "window", pyWindowDict);
    break;
}

if (PyDict_Size(pyDict) > 0)
    return pyDict;

return Py_BuildValue(""); /* otherwise None */
```


B Viewpoint tracker implementation

```c
static PyObject* getType(PyObject *self, PyObject *args) {
    PyObject *obj;
    XAnyEvent *xevent;
    if (!PyArg_ParseTuple(args, "O:get_type", &obj))
        return NULL;
    xevent = (XAnyEvent *)PyCObject_AsVoidPtr(obj);
    return Py_BuildValue("i", xevent->type);
}

// return the pointing error of an event as difference between the center
// of the event window and the pointing position
// considering button press and release events
static PyObject* getPointingError(PyObject *self, PyObject *args) {
    PyObject *pyCobj;
    XAnyEvent *xEvent;
    int errorX, errorY;
    // parse arguments
    if (!PyArg_ParseTuple(args, "O:get_type", &pyCobj))
        return NULL;
    xEvent = (XAnyEvent *)PyCObject_AsVoidPtr(pyCobj); // extract xevent from
    PyObject *pyCobj;
}

// apply position correction
static PyObject* applyPositionCorrection(PyObject *self, PyObject *args) {
    PyObject *pyCobj; /* python’s void pointer ref of the x11 event */
    XAnyEvent *xEvent;
    int xDiff, yDiff; /* correction parameter */
    /* check arguments */
    if (!PyArg_ParseTuple(args, "Oii", &pyCobj,&xDiff,&yDiff))
        return NULL;
    xEvent = (XAnyEvent *)PyCObject_AsVoidPtr(pyCobj); // extract xevent from
    PyObject *pyCobj;
}

// examine window
static PyObject* _examineWindow(XAnyEvent *xEvent) {
// return value
    PyObject *pyWindowDict;
    pyWindowDict = PyDict_New();
    // examine window
    int x, y;
```
unsigned int width, height, bw, depth, nchildren_return;
Window root_return, parent_return, *children_return;

int status;

status = XGetGeometry(xEvent->display, xEvent->window, &root_return, &x, &y, &width, &height, &bw, &depth);
if (status > 0) {
    PyDict_SetItemString(pyWindowDict, "x", Py_BuildValue("i", x));
    PyDict_SetItemString(pyWindowDict, "y", Py_BuildValue("i", y));
    PyDict_SetItemString(pyWindowDict, "width", Py_BuildValue("i", width));
    PyDict_SetItemString(pyWindowDict, "height", Py_BuildValue("i", height));
    PyDict_SetItemString(pyWindowDict, "bw", Py_BuildValue("i", bw));
    PyDict_SetItemString(pyWindowDict, "depth", Py_BuildValue("i", depth));
} else {
    printf("Error in XGetGeometry()\n");
}

status = XQueryTree(xEvent->display, xEvent->window, &root_return, &parent_return, &children_return, &nchildren_return);
if (status > 0) {
    PyDict_SetItemString(pyWindowDict, "children", Py_BuildValue("i", nchildren_return));
} else {
    printf("Error in XQueryTree()\n");
}

return pyWindowDict;

static PyMethodDef module_functions[] = {
    {"getType", getType, METH_VARARGS, "returns the type of event as integer."},
    {"getEventAsPyDict", getEventAsPyDict, METH_VARARGS, "returns event info as a python dictionary of x11 button and movement event."},
    {"applyPositionCorrection", applyPositionCorrection, METH_VARARGS, "apply x,y correction to the pointing position of the event."},
    {"getPointingError", getPointingError, METH_VARARGS, "get pointing error [ diffX, diffY] relative to the center of the event window"},
    { NULL }
};

void initPyXevent()
{
    (void) Py_InitModule3("PyXevent", module_functions, "X11 Event module");
}

Listing B.11: Xevent header

/*******************************************************************************/
X11 Event module

Copyright 2010 by Bastian Migge <miggeb@ethz.ch>

License

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2 of the License, or (at your option) any later version.
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Links

http://nedbatchelder.com/text/whirlext.html
http://docs.python.org/release/1.5.2/api/
http://docs.python.org/release/2.5.2/api/cObjects.html

天涯

### Listing B.12: Xevent source

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <X11/Xlib.h>

// get window (grandchild of given one) at root position
Window findWindowAt(Display *display, Window window, int *x_root, int *y_root);

// return the pointing error of an event relative to it’s window center
int _getPointingError(XAnyEvent *xEvent, int *xError, int *yError);

// apply correction parameter (xDiff,yDiff) to the pointer position of the x11 event
int _applyPositionCorrection(XAnyEvent *xEvent, int xDiff, int yDiff);
```

X11 Event module

Copyright 2010 by Bastian Migge <miggeb@ethz.ch>

License

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Links

http://nedbatchelder.com/text/whirlext.html
http://docs.python.org/release/1.5.2/api/
http://docs.python.org/release/2.5.2/api/cObjects.html
Window findWindowAt(Display *display, Window window, int *x_root, int *y_root)
{
    Window root_return, parent_return, children_return;
    int childIndex, found;
    unsigned int nchildren_return;
    Window ret_val = window;

    do {
        found = 0;
        if (XQueryTree(display, ret_val, &root_return, &parent_return, &children_return, &nchildren_return))
        {
            for (childIndex = 0; childIndex < nchildren_return; ++childIndex)
            {
                Window root;
                int x, y;
                unsigned int width, height, bw, depth;

                XGetGeometry(display, children_return[childIndex], &root, &x, &y, &width, &height, &bw, &depth);

                // if (x_root, y_root) is within child
                if ((x_root >= x && x_root <= x + width && y_root >= y && y_root <= y + height)
                {
                    ret_val = children_return[childIndex];
                    found = 1;
                    break;
                }
            }
        }
    } while (found == 1);

    if (found == 1)
    {
        XFree(children_return);
        return ret_val;
    }
    else
    {
        XFree(children_return);
        return 0;
    }
}

int _getPointingError(XAnyEvent *xEvent, int *xError, int *yError)
{
    XButtonEvent *xButtonEvent;
    Window root;
    unsigned int width, height, bw, depth, centerX, centerY;
    int x, y;

    if (xEvent->type == ButtonPress || xEvent->type == ButtonRelease)
    {
        xButtonEvent = (XButtonEvent *) xEvent;
        xButtonEvent.
    }
}
B Viewpoint tracker implementation

```c
XGetGeometry(xButtonEvent->display, xButtonEvent->window, &root, &x, &y, &width, &height, &bw, &depth);
centerX = x+width/2;
centerY = y+height/2;
*xError = xButtonEvent->x - centerX;
*yError = xButtonEvent->y - centerY;
return 0;
}
return 1;
}

int _applyPositionCorrection(XAnyEvent *xEvent, int xDiff, int yDiff)
{
    XButtonEvent *xButtonEvent;
    XMotionEvent *xMotionEvent;
    Window newWindow;  /* window update for corrected position */

    switch (xEvent->type) {
    case ButtonPress:
        case ButtonRelease:
        xButtonEvent = (XButtonEvent *) xEvent;
        // update position
        xButtonEvent->x += xDiff;
        xButtonEvent->y += yDiff;
        xButtonEvent->x_root += xDiff;
        xButtonEvent->y_root += yDiff;
        /* update 'event' window, root and subwindow are not updated so fare */
        newWindow = findWindowAt(xButtonEvent->display,xButtonEvent->root,&
            xButtonEvent->x_root,&xButtonEvent->y_root);
        xButtonEvent->window = newWindow;
        return 0;
    case MotionNotify:
        xMotionEvent = (XMotionEvent *) xEvent;
        // update position
        xMotionEvent->x += xDiff;
        xMotionEvent->y += yDiff;
        xMotionEvent->x_root += xDiff;
        xMotionEvent->y_root += yDiff;
        /* update 'event' window, root and subwindow are not updated so fare */
        newWindow = findWindowAt(xMotionEvent->display,xMotionEvent->root,&
            xMotionEvent->x_root,&xMotionEvent->y_root);
        xMotionEvent->window = newWindow;
        return 0;
    default:
        return 1;
    }
}
```

Preprocessor

As introduced in Section 5.7.4, the preprocessor calculates the interaction error for a given interaction position and target (GUI widget).
B.2 Parallax error correction controller prototype

The error is calculated as offset between interaction position and target focal point. Since the target point is mostly the geometrical center of the target, it can be deduced from the position \( p_{x,y} \) and the geometry \( h, w \) of the target. It is given in 2-dimensional display coordinates. Additionally, it is normalized according to the POMDP model unit.

**Listing B.13: X11 Preprocessor**

```python
# -*- coding: utf-8 -*-

pyqtpc.gui.pyqt4.app.x11

from pyqtpc.gui.pyqt4.app import QApplicationPointingErrorCorrected, QApplicationPointingErrorCorrectedSimulation
from pyqtpc.gui.x11.x11Event import PyXevent
from pyqtpc.correction.control import CorrectionControllerIOSenseData

class QApplicationPointingErrorCorrectedX11(QApplicationPointingErrorCorrected):
    ""
    Pointing Corrected QApplication for X11
    ""

def x11EventFilter(self, event):
    ""
    X11 event callback on application level. Every event from the GUI passes this point.
    * extract event information to pyXevent dict and forward event to super.
    * ask correction controller for correction parameter
    * apply corrected pointer position to x11Event event
    * generate and send observation (click error with correction, target) to controller
    * pass event to window
    ""
    pyEvent = PyXevent.getEventAsPyDict(cObject)
```

---

207
B Viewpoint tracker implementation

```python
# filter event
if not self.validEvent(pyEvent):
    return False

correction_offset = self.get_correction(pyEvent, self.
correction_controller)

# apply latest correction from controller to X11 interaction event, which
# will be passed to the GUI then.
PyXevent.applyPositionCorrection(cObject,correction_offset[0],
correction_offset[1]) # xevent

# ignore event in any case, such that the app passes it to the active
# widget
# !!! DON’T CHANGE THIS !!!
return False
# !!! DON’T CHANGE THIS !!!

class QApplicationPointingErrorCorrectedX11Simulation(QApplicationPointingErrorCorrectedX11,
QApplicationPointingErrorCorrectedSimulation):
    ""
    Pointing Corrected QApplication for X11
    ""

def x11EventFilter(self, event):
    ""
    X11 event callback on application level
    extract event information to pyXevent dict and forward event to super.
    processInteractionEvent()
    
    Annotation
    QApplication.x11ProcessEvent can not be overwritten in PyQt
    Arguments
    event - X11 event (pointer to C struct see Xlib manual)
    ""

    # extract native x11Event event in C to python dict
    cObject = event.ascobject() # http://docs.python.org/release/2.5.2/api/
cObjects.html
    pyEvent = PyXevent.getEventAsPyDict(cObject)

    QApplicationPointingErrorCorrectedSimulation.logInteractionEvent(self,
    pyEvent)

    return QApplicationPointingErrorCorrectedX11.x11EventFilter(self, event)
```

Webbrowser application

Listing B.14: X11 Preprocessor

```bash
# -*- coding: utf-8 -*-
""
pygtpc.gui.pyqt4.widget.webview
~~~~~~
```
import sys
from PyQt4 import QtGui, QtCore, QtWebKit
from pyqtpc.gui import RealWorldFeedbackInterface

class QWebWindow(QtGui.QMainWindow):
    
    A simple webbrowser with status bar, forward & back button, url bar and
    some keyboard shortcuts

    Keyboard Shortcuts:
    Ctrl+F -- find
    Ctrl+Q -- Quit
    Ctrl+{-/+} -- Zoom

    based on https://code.google.com/p/devicenzo/

    def __init__(self, url=QtCore.QUrl('http://www.python.org')):
        QtGui.QMainWindow.__init__(self)
        self.sb=self.statusBar()

        self.pbar = QtGui.QProgressbar()
        self.pbar.setMaximumWidth(120)
        self.wb=QtWebKit.QWebView(loadProgress = self.pbar.setValue, loadFinished = self.pbar.hide, loadStarted = self.pbar.show, titleChanged = self.setWindowTitle)
        self.setCentralWidget(self.wb)

        self.tb=self.addToolBar("Main Toolbar")
        for a in (QtWebKit.QWebPage.Back, QtWebKit.QWebPage.Forward, QtWebKit.QWebPage.Reload):
            self.tb.addAction(self.wb.pageAction(a))

        self.url = QtGui.QLineEdit(returnPressed = lambda: self.wb.setUrl(QtCore.QUrl.fromUserInput(self.url.text())))
        self.tb.addWidget(self.url)

        self.wb.urlChanged.connect(lambda u: self.url.setText(u.toString()))

        self.wb.urlChanged.connect(lambda: self.url.setCompleter(QtGui.QCompleter([QtCore.QString(i.url().toString()) for i in self.wb.history().items()], caseSensitivity = QtCore.Qt.CaseInsensitive))

        self.wb.statusBarMessage.connect(self.sb.showMessage)

        self.wb.page().linkHovered.connect(lambda l: self.sb.showMessage(l, 3000))

        self.search = QtGui.QLineEdit(returnPressed = lambda: self.wb.findText(self.search.text()))
        self.search.hide()

        self.showSearch = QtGui.QShortcut("Ctrl+F", self, activated = lambda: (self.search.show() , self.search.setFocus()))
        self.hideSearch = QtGui.QShortcut("Esc", self, activated = lambda: (self.search.hide(), self.wb.setFocus()))
B Viewpoint tracker implementation

```python
self.quit = QtGui.QShortcut("Ctrl+Q", self, activated = self.close)
self.zoomIn = QtGui.QShortcut("Ctrl++", self, activated = lambda: self.wb.setZoomFactor(self.wb.zoomFactor()+.2))
self.zoomOut = QtGui.QShortcut("Ctrl+-", self, activated = lambda: self.wb.setZoomFactor(self.wb.zoomFactor()-0.2))
self.zoomOne = QtGui.QShortcut("Ctrl+=", self, activated = lambda: self.wb.setZoomFactor(1))
self.wb.settings().setAttribute(QtWebKit.QWebSettings.PluginsEnabled, True)

self.sb.addPermanentWidget(self.search)
self.sb.addPermanentWidget(self.pbar)
self.wb.load(url)

class QWebWindowWithFeedback(QWebWindow, RealWorldFeedbackInterface):
    
    Implementation of Parallax correction Window for QWebView (HTML browser)

    see 'CorrectedWindow' description for details
    
    def __init__(self, *args, **kwargs):
        super(QWebWindowWithFeedback, self).__init__(*args, **kwargs)
        #QWebView.__init__(self, *args, **kwargs)
        self.validTargetClasses = [QtWebKit.QWebElement]

def keyPressEvent(self, e):
    if e.key() == QtCore.Qt.Key_Escape:
        self.close()

def targetAt(self, position, frame=None):
    
    return first hyper link (<a href>) web elements at global position

    arguments:
        interaction_position -- interaction position in display coordinates
            as python list
        frame -- QFrame to search in (default: page.mainframe())
    
    position_Qt = QtCore.QPoint(position[0], position[1])
    if (frame == None):
        frame = self.wb.page().mainFrame()
    collection = frame.findAllElements("a")
    for elementIndex in range(collection.count()):
        element = collection.at(elementIndex)
        # find hyper links
        href = element.attribute("href")
        if (not href.isEmpty()):
            geometryLocal = element.geometry()
            positionLocal = self.mapFromGlobal(position_Qt)
            #print geometryLocal, positionLocal
```

B.2 Parallax error correction controller prototype

```python
if (geometryLocal.contains(positionLocal)):
    if isinstance(element, QtWebKit.QWebFrame):
        return self.targetAt(position_Qt, frame = element)
    else:
        return element
return self.wb.page().mainFrame()

def targetFieldOfAttention(self, target):
    ""
    return the field of attention of a QWidget in global coordinates
    arguments
    target - [QFrame]
    ""
    geometry = target.geometry()
    center_Qt = QtCore.QPoint(geometry.x() + geometry.width()/2., geometry.y()
                        + geometry.height()/2.)
    center_global_Qt = self.mapToGlobal(center_Qt)
    center = [center_global_Qt.x(),center_global_Qt.y()]
    return center

def targetSize(self, target):
    ""
    returns target size as python list
    arguments:
    target - [QFrame]
    ""
    geom = target.geometry()
    size = [geom.width(),geom.height()]
    return size
```

if __name__ == "__main__":
    app=QtGui.QApplication(sys.argv)
    if len(sys.argv) > 1:
        url = QtCore.QUrl.fromUserInput(sys.argv[1])
    else:
        url = QtCore.QUrl('http://www.python.org')
    wb=QWebWindow(url)
    wb.show()
    sys.exit(app.exec_())

B.2.3 POMDP model

The following listing shows the POMDP model for correcting the horizontal parallax error.

```plaintext
# POMDP model for parallax error correction
# copyright 2010,2011,2012,2013 by Bastian Migge <miggeb@ethz.ch>
# the model is deduced from studying the user behavior in front
# of interactive screens to correct the parallax error.

discount: 0.99
values: cost

# actions
```

Listing B.15: Parallax error POMDP model (Cassandra format)
B Viewpoint tracker implementation

actions: 3
# 0 - shift vp left, increase correction
# 1 - do not change vp assumption, no correction change
# 2 - shift vp right, decrease correction

# states
states: 5

# observations
observations: 5

# observation model S x O = P(S|O)
O:* 1.0 0.0 0.0 0.0 0.0
0.64878052 0.35121948 0.0 0.0 0.0
0.28358209 0.35820895 0.35820896 0.0 0.0
0.0 0.0 0.64285715 0.32142856 0.03571429
0.0 0.0 0.85714286 0.14285714 0.0

#transition model P(S'|S,A)
T:1 # no shift
0.84821429 0.14285714 0.00892857 0.0 0.0
0.17410715 0.67410714 0.14285714 0.00892857 0.0
0.01339286 0.16071429 0.67410714 0.14285714 0.00892857
0.0 0.01339286 0.16071429 0.67410714 0.15178571
0.0 0.0 0.01339286 0.16071429 0.82589285

T:0 # shift vp right (decrease correction)
0.17410715 0.67410714 0.14285714 0.00892857 0.0
0.01339286 0.16071429 0.67410714 0.14285714 0.00892857
0.0 0.01339286 0.16071429 0.67410714 0.15178571
0.0 0.0 0.01339286 0.16071429 0.82589285
0.0 0.0 0.0 0.01339286 0.98660714

T:2 # shift vp left (increase correction)
0.99107143 0.00892857 0.0 0.0 0.0
0.84821429 0.14285714 0.00892857 0.0 0.0
0.17410715 0.67410714 0.14285714 0.00892857 0.0
0.01339286 0.16071429 0.67410714 0.14285714 0.00892857
0.0 0.01339286 0.16071429 0.67410714 0.15178571

#reward model R: <action> : <start-state> : <end-state> : <observation>
# since the controller changes the setup, the user is irritated -> costs 1
# the system state also costs: error state 0,2 -> costs 10

# increase correction
R:0:*:0:* 11.0 # wrong state + correction changed
R:0:*:1:* 11.0 # wrong state + correction changed
R:0:*:2:* 1.0 # right state + correction changed
R:0:*:3:* 11.0 # wrong state + correction changed
R:0:*:4:* 11.0 # wrong state + correction changed

# do nothing
R:1:*:0:* 10.0 # wrong state + no user irritation
R:1:*:1:* 10.0 # wrong state + no user irritation
R:1:*:2:* 0.0 # right state + no user irritation
R:1:*:3:* 10.0 # wrong state + no user irritation
R:1:*:4:* 10.0 # wrong state + no user irritation
B.2 Parallax error correction controller prototype

# decrease correction
R:2:*:0:* 11.0 # wrong state + correction changed
R:2:*:1:* 11.0 # wrong state + correction changed
R:2:*:2:* 1.0 # right state + correction changed
R:2:*:3:* 11.0 # wrong state + correction changed
R:2:*:4:* 11.0 # wrong state + correction changed
Waste to energy plant model

This section shows the waste to energy plant implementation in MATLAB/Simulink®. Figure C.1 shows the water steam cycle. It is set up according to the clausius-rankine cycle. To control the heating transfer from the boiler, which is heated by the flue gas of the combustion, to the electricity grid and the district heating, bypasses are implemented. The heating unit of the boiler can be bypassed, as shown in Figure C.2. It represents to inject liquid water after the super heating to cool the steam and reduce its energy. The turbine bypass (value) allows to directly feed the condenser instead of relaxing the steam by the turbine and the district heating heat exchanger. The district heating split value (3-way-valve), allows to lead the either to the second turbine stage or to the district heating.
Figure C.1: Water steam process (MATLAB/Simulink®)

Figure C.2: Boiler with bypass (MATLAB/Simulink®)
PyMDP is a Python [15] library for working with MDPs and Reinforcement learning written by Oliver Stollmann and Bastian Migge. The toolbox provides a framework for developing planning and learning algorithms as well as system simulation. The software provides the following features: Model and control policy representation, controller, planner, learner and simulator.

MDP and POMDP models represent the behavior of controlled systems under uncertainty and a cost function. PyMDP provides object-oriented data structures for representing such models and control policies as well as serialization and deserialization from and to the Cassandra-POMDP-format (see Section 3.2.2) and JSON. (JSON - JavaScript Object Notation is a text-based open standard designed for human-readable data interchange.)

Moreover, the framework provides algorithms for MDP and POMDP control, planning and reinforcement learning. It implements dynamic programming solvers for finite horizon and discounted infinite horizon planning problems, such as value iteration and policy iteration (see Section 2.4.3 and 2.5.4) and genetic algorithm solver. Controller implement a control loop to sequentially execute control policies on a system, which represents a real system or a simulation. (The system interface is discussed below.)

To automatically deduce policies from existing environments (or simulations), the toolbox provides the reinforcement learning implementation for monte carlo value estimation, incremental TD(0) policy evaluation, Incremental sarsa- as well as q-learning and several exploration/exploitation strategies, i.e. greedy, $\epsilon$-greedy, softmax and random policy generators. To extract MDPs models from existing simulations, a frequency counting MDP extractor is implemented.

To test controllers and learners, the toolbox provides system simulation. Based on a given system model, system simulators allow to sample the probabilistic system behavior. The system simulator interface is shown in Listing D.1. It provides loading the simulation with a system representation object (model) and executing an action at the system. This affects the current system state according to the model and returns the gathered reward.

Listing D.1: PyMDP system interface

```python
class Simulator(object):
    ''' Simulator interface '''

    def __init__(self, mdp, init_state):
        pass

    def state(self):
        ''' current system state '''
        pass
```
D pyMPD: Markov decision process toolbox

```python
def reward(self):
    ''' return last reward '''
    pass

def act(self, action):
    ''' simulate one time step
    :param action: action id
    :return: reward
    '''
    pass
```

PySimulink: MATLAB/Simulink® simulation interface

In Section D, the library containing controller and learner algorithms and the system interface is presented. However, to evaluate controllers on realistic models and learn from complex models, the system interfaces must provide a mechanism to connect the PyMDP framework, which is written in Python, and an external simulation software, i.e. written in MATLAB/Simulink® [8]. MATLAB/Simulink® offers a number of options regarding connectivity to outside controllers, which can be grouped into live communications and simulation management.

**Live communications** allows external controllers to communicate with a running Simulink model using hardware and software tools, such as Simulink Real-Time Windows Target[18] or the OPC toolbox[10]. Real-Time Windows Target allows to integrate hardware I/O boards in order to control real-time systems for rapid prototyping or hardware-in-the-loop simulation. This enables developing software that will be used to communicate with real systems instead of simulations. Object Linking and Embedding for Process Control (OPC) is a more general approach for communication in industrial automation. It is a standard specification for exchanging real-time plant data between control devices from different manufacturers. The OPC toolbox for Simulink allows to run simulations instead of real plants. The main disadvantage of this implementations is the limitation to Microsoft Windows running machines. The needed OPC Data Access Specification, for instance, depends on Microsoft’s Distributed Component Object Model (DCOM) technology for inter process communication, which is proprietary software.

An alternative approach is simulation management. It is chosen if the communication between controller and simulation is implemented within the MATLAB environment. A later switch to a real system is not easily possible, since the interfaces must be adapted. In contrast to live communications, simulation management does not run a controller on a live system simulation. Instead, simulations are sequentially started, stopped and restarted. Between the simulation runs, a simulation manager reads the outputs of the previous simulation and produces input for the following run, i.e. the initial system state and control actions. The approach offers more flexibility than live communications, since it moves timing responsibility from the model solver to the simulation manager. It simplifies the simulation clocking control compared to live communications.

The second option, simulation management, is the selected choice for PySimulink in order to integrate PyMDP and MATLAB/Simulink®, since providing a qualitative performance evaluation and the non-hardware-in-the-loop nature of the given problem (see Section 6.2.6) does not warrant the effort of live communications.

**PySimulink** is a simulation management written in Python and MATLAB. The communication between controller and simulation is a file-based protocol in JSON. These libraries encapsulate the definition of the simulation start- and end-time, inputs and outputs and the system state. The simulation management
hides these parameters, such that the MATLAB/Simulink simulation provides the same simple interface to the controller, shown in Listing D.1, as any other system representation. Thus, the framework offers not only a intuitive but also an immediately usable system simulation with the existing data structures and algorithms presented in Section D.

Figure D.1 shows an overview in the pySimulink software structure. The software works as follows. A controller, as introduced in Section D, interacts with the Python simulation manager though the common interface in Python, defined in Listing D.1. The controller applies actions to the simulation and gets the new system state as well the immediate reward in return. The return value is deduced from the Simulink simulation as follows. Based on the current system state and the controllers action, the Python simulation manager sets up a Simulation request and sends it to the Matlab simulation manager, which is written in MATLAB. The Matlab simulation manager deserializes the request and starts the MATLAB/Simulink simulation accordingly. After the simulation is finished, the results (time series of outputs and final state of the system) are formulated as Simulation response and send back to the Python simulation manager. The results are deserializes and send as response (system state and reward) to the controller.

The information exchange (simulation request and response) is file based in the JSON format, which are stored in specified folders. Request are placed in the incoming/ and responses are stored in the outgoing/ directory. Both simulation manager, the one in Python and the one written in MATLAB, must have access to the folders. Due to network file sharing it is possible to set up the controller and the simulator on different machine. Since the exchanged files are expressed in the JSON format, the software is also independent of the operating system.

Figure D.1 provides an overview of the PySimulink simulation object towards the controller or learner. It assumes, that the simulation directory (defined by the variable sim_dir) has two subfolders: The incoming/ and outgoing/
and the outgoing/ folder. The Simulink model is assumed to be located in the simulation directory with the filename given by model.

**Listing D.2: PySimulink system configuration**

```python
code
system = pysim.SimulationSystem(
    host_os = 'win',
    model = 'WtEPlant',
    sim_wait_time = 100,
    steps_per_sim = 180,
    state_signal_name = 'state_id',
    action_signal_name = 'action_id',
    reward_signal_name = 'reward',
    sim_dir = '/mnt/wte/MATLAB/',
    states_dir = r'D:\wte\MATLAB\states')
```

According to the simulation interface (see Listing D.1), the `system` variable would now offer three methods, `act()`, `state()` and `reward()`, which can be used to trigger simulations or request the current state and reward of the simulated system. The `act()` method produces a simulation request file, as described above, and lets the **Python simulation manager** wait for the simulation response for the time of `sim_wait_time [s]`. The **Matlab simulation manager**, once started by the `run()` method, continuously waits for incoming files monitoring the `incoming/` directory and starts a simulation once a new file is found. It will use the read parameters to instruct Simulink to load initial states and store final state. Since the state and the action definition defined by the controller is abstract, the extensive state definition is stored in the `states_dir` directory using the **MATLAB Simstate** feature.
Figure D.2: *pySimulink*: MATLAB/Simulink® components
Mathematical notation

E.1 Probability theory

Discrete random variable

Let \( x \) be a discrete random variable over the discrete space \( S = \{ s_1, \ldots, s_{|S|} \} \) with length \( |S| \) \[77\]. Then the \( i \)-th element of the stochastic vector \( b \), \( b_i \) denotes the probability that \( x \) assumes the value \( s_i \):

\[
b_i = P(x = s_i), \quad i = 1, \ldots, |S|.
\] (E.1)

Then, \( b_i \) must satisfy the following conditions:

\[
b_i \in [0, 1] \quad \text{and} \quad \sum_{i=1}^{|S|} b_i = 1
\] (E.2)

A discrete random variable is expressed as probability vector, stochastic vector or distribution vector, for example the POMDP belief state \((b)\) as probability distribution over the underlying MDP (domain) states.

\[
P(x|y)
\] (E.3)

E.2 Stochastic matrix

A stochastic matrix describes the transitions of a Markov chain over a finite state space \( S \). Each of its entries is real number between 0 and 1, representing a probability. Either each row or column is summing to 1.

For \( s_i \in S, i \in \{0, \ldots, |S| - 1\} \) the transition is given by \( P(s_j|s_i) = P_{i,j} \) with

\[
P = \begin{pmatrix}
P(s_1|s_0) & \cdots & P(s_1|s_{|S|-1}) \\
\vdots & \ddots & \vdots \\
P(s_{|S|-1}|s_0) & \cdots & P(s_{|S|-1}|s_{|S|-1})
\end{pmatrix}
\] (E.4)
E Mathematical notation

It is also termed as **probability matrix** or **transition matrix**. Examples for **stochastic matrices** with additional conditions are used to express the MDP transition model, which represents a Markov chain for a given action $a_k$ as tensor $P_{i,j,k} = P(s_j|s_i, a_k)$ and the POMDP observation model $P(s|o, a)$ as introduced in Section 2. Transition and observation models are expressed as probability matrices $P$.

A *stationary* stochastic vector $\vec{b}$ is defined as vector that does not change under the application of the stochastic matrix $P$:

$$\vec{b}P = \vec{b} \quad \text{(E.5)}$$

A stochastic matrix (with strictly positive entries) has a unique stationary stochastic vector $\vec{b}$ [171]:

$$\lim_{t \to \infty} P^{(t)}b = \vec{b} \quad \text{(E.6)}$$

This implies, that the convergence vector $\vec{b}$ of $P$ is independent of the initial probability $b$.

In the context of MDPs, $\vec{b}$ is called steady state of the system, since $P$ models the system dynamics (over time $t$) within one time step. And $\lim_{t \to \infty} P^{(t)}$ denotes the long-term state of the system which is interpreted as unique, steady belief state $\vec{b}$ [171].
## List of Figures

2.1 Sequential decision process .............................................. 7  
2.2 Accumulated cost function of model predictive control ............. 11  
2.3 Markov decision process for two actions \( a_0, a_1 \) .................. 14  
2.4 Forward search and value back propagation over two stages (and-or tree) 17  
2.5 Hidden Markov model estimation process .............................. 18  
2.6 Value function of MDP with maximum likelihood estimator ........... 21  
2.7 Policy tree ...................................................................... 23  
2.8 POMDP value function ...................................................... 25  
2.9 Value functions for estimated MDP and POMDP ......................... 27  
4.1 Symbolic states of real world values ...................................... 41  
4.2 POMDP model element dependencies .................................... 43  
4.3 Manual modeling process .................................................. 43  
4.4 Automated modeling process ............................................ 44  
4.5 Simulation system interface ............................................... 44  
4.6 Controlled estimator ...................................................... 47  
4.7 Convergence of the oblivion process ..................................... 52  
5.1 Human-computer interaction on interactive surfaces .................... 55  
5.2 2-dimensional human computer interface setups ....................... 57  
5.3 User awareness on interactive screens .................................... 58  
5.4 Offset between interaction and image plane ............................ 61  
5.5 Robust interactive screens ................................................ 62  
5.6 Geometrical dependence of the interaction point on the viewpoint 64  
5.7 Interaction error and dexterity [mm] (histogram) ....................... 65  
5.8 Static display calibration .................................................. 70  
5.9 Static calibration bias ..................................................... 71  
5.10 Offset between the image plane and the interaction plane [180] .... 72  
5.11 Interactive screens in public areas ...................................... 73  
5.12 Measured values (schematic) ............................................ 79  
5.13 User viewpoint position relative to the screen [mm] (histogram) .... 80  
5.14 User viewpoint distance to screen [mm] (histogram) .................. 80  
5.15 Offset between viewpoint and interaction point [mm] (histogram) 80  
5.16 User movement \( \frac{mm}{1.32s} \) (histogram) ................................. 84  
5.17 Dexterity model ............................................................ 85  
5.18 Frequency of the relative viewpoint and interaction error (thermal plot) 86  
5.19 Normalized relative viewpoint and interaction error (scatter plot) 86
List of Figures

5.20 Smoothed correlation between relative viewpoint and interaction error .......................... 87
5.21 Interaction error with bare ginger and pen TUI [mm] (histogram) ................................. 88
5.22 Detection zone of small area targets (widgets) ............................................................. 89
5.23 Geometrical center of GUI widgets .................................................................................. 89
5.24 Optical center of GUI widgets ........................................................................................ 90
5.25 Influence of optical center onto interaction error for buttons ....................................... 91
5.26 Interaction error on different widget classes .................................................................... 91
5.27 Correlation between stability of horizontal interaction error and button size for buttons .... 92
5.28 Correlation between stability of horizontal interaction error and target size for links (height 18px) .............................................................................................................. 92
5.29 Tracking based parallax error correction ........................................................................ 95
5.30 Vision based user tracking (depth sensor image) ............................................................. 96
5.31 Viewpoint tracking filter .................................................................................................. 97
5.32 Deviation space ................................................................................................................ 100
5.33 Parallax error correction by viewpoint estimation ............................................................ 100
5.34 Observation model for a given viewpoint and target as discrete probability distribution .... 101
5.35 State space reduction ...................................................................................................... 103
5.36 Discrete observation model ............................................................................................. 105
5.37 State machine of parallax error correction controller ..................................................... 107
5.38 POMDP controller system components ......................................................................... 110
5.39 Correction of a static interaction error ............................................................................ 113
5.40 Correction of oscillating interaction error with small amplitude ................................... 113
5.41 Correction of oscillating interaction error with large amplitude ..................................... 114

6.1 Energy flow of a waste incineration plant ......................................................................... 120
6.2 Water steam cycle (schematic) ......................................................................................... 121
6.3 Drum control (schematic) ................................................................................................ 124
6.4 Steam outlet control (schematic) ..................................................................................... 125
6.5 Heating control (schematic) .............................................................................................. 126
6.6 Feedback control loop ....................................................................................................... 131
6.7 Waste to energy plant ’KVA Turgi’ (schematic) ............................................................... 135
6.8 Bunker component (MATLAB/Simulink®) ................................................................. 138
6.9 Combustion component (MATLAB/Simulink®) ............................................................ 138
6.10 Water steam cycle control (schematic) .......................................................................... 139
6.11 Waste component (MATLAB/Simulink®) .................................................................. 140
6.12 Energy sales pricing model ............................................................................................. 141
6.13 Heat exchanger step responses (Thermolib) ................................................................. 144
6.14 Energy production step responses .................................................................................. 144
6.15 Waste to energy plant decision points ........................................................................... 145
6.16 Reinforcement learning interface ................................................................................... 147
6.17 Q-value history for state 74 with $\alpha = 0.5$, $\gamma = 0.5$ ......................................... 151
6.18 Q-value history for state 74 with $\alpha = 0.7$, $\gamma = 0.7$ ............................................. 152
6.19 Q-value history for state 74 with $\alpha = 0.9$, $\gamma = 0.9$ ............................................. 152
6.20 Policy evaluation by accumulated reward of controller application ............................. 154

A.1 Hardware setup of the interactive screen ’DigiBench’ ....................................................... 159
A.2 User study system setup .................................................................................................. 160
A.3 Large interactive system with 3-dimensional head tracking ........................................... 161
A.4 Interaction position on screen (scatter plot and histogram) ........................................... 163
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.5</td>
<td>3-dimensional head tracking</td>
<td>165</td>
</tr>
<tr>
<td>A.6</td>
<td>GUI application simulating common targets on the screen (screenshot)</td>
<td>165</td>
</tr>
<tr>
<td>A.7</td>
<td>User viewpoint location [mm] (histogram)</td>
<td>166</td>
</tr>
<tr>
<td>A.8</td>
<td>User viewpoint movement with 0.2 Hz [mm] (histogram)</td>
<td>166</td>
</tr>
<tr>
<td>A.9</td>
<td>User viewpoint movement with 1 Hz [mm] (histogram)</td>
<td>166</td>
</tr>
<tr>
<td>A.10</td>
<td>GUI button</td>
<td>170</td>
</tr>
<tr>
<td>A.11</td>
<td>GUI checkboxes (Qt)</td>
<td>170</td>
</tr>
<tr>
<td>A.12</td>
<td>Interactive system setup</td>
<td>171</td>
</tr>
<tr>
<td>A.13</td>
<td>Application GUIs</td>
<td>172</td>
</tr>
<tr>
<td>A.14</td>
<td>Aimed interaction position (questionnaire)</td>
<td>173</td>
</tr>
<tr>
<td>B.1</td>
<td>GUI structure</td>
<td>197</td>
</tr>
<tr>
<td>B.2</td>
<td>X11 architecture</td>
<td>198</td>
</tr>
<tr>
<td>C.1</td>
<td>Water steam process (MATLAB/Simulink®)</td>
<td>216</td>
</tr>
<tr>
<td>C.2</td>
<td>Boiler with bypass (MATLAB/Simulink®)</td>
<td>216</td>
</tr>
<tr>
<td>D.1</td>
<td>Python/MATLAB simulation management (components and message passing)</td>
<td>219</td>
</tr>
<tr>
<td>D.2</td>
<td>pySimulink: MATLAB/Simulink® components</td>
<td>221</td>
</tr>
</tbody>
</table>
# List of Tables

1.1 Synonymous terms of control systems [52][184] ................................. 4
2.1 Markov models .................................................................................. 8
5.1 Interaction error and dexterity characteristics [mm] ............................ 66
5.2 Hit rate with and without correction [%] ........................................... 66
5.3 Viewpoint characteristics [mm] ......................................................... 81
5.4 Horizontal viewpoint characteristics relative to the target [mm] ........ 82
5.5 Viewpoint movement characteristics [mm] ........................................ 83
5.6 Interaction error with bare finger and pen TUI [mm] ......................... 88
5.7 Anderson-Darling test for normal distribution ................................. 92
5.8 Aimed interaction position .............................................................. 92
6.1 Model parameter verification against 'KV A Turgi' ........................... 142
6.2 Waste fraction [40][39] ................................................................... 143
6.3 Waste heating value [40][39] ............................................................. 143
6.4 State discretization .......................................................................... 149
6.5 Action discretization ....................................................................... 149
6.6 I/O interface and model discretization ............................................ 149
Bibliography


Bibliography


1999.


Bibliography


