Modelling geometric uncertainties in robotic assembly

Author[s]:
Müller, Markus Andreas

Publication Date:
1993

Permanent Link:
https://doi.org/10.3929/ethz-a-000697620

Rights / License:
In Copyright - Non-Commercial Use Permitted
MODELING GEOMETRIC UNCERTAINTIES IN ROBOTIC ASSEMBLY

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH
for the degree of
Doctor of Technical Sciences

presented by
MARKUS ANDREAS MÜLLER
Dipl. El.-Ing. ETH
born October 8, 1961
citizen of Basel and Embrach,
Switzerland

accepted on the recommendation of
Prof. Dr. M. Mansour, examiner
Prof. Dr. W. Schaufelberger, co-examiner
Dr. E. Badreddin, co-examiner

1993
'We have sailed many months, we have sailed many weeks
   (Four weeks to the month you may mark),
But never as yet ('tis your Captain who speaks)
   Have we caught the least glimpse of a Snark!
'We have sailed many weeks, we have sailed many days
   (Seven days to the week I allow),
But a Snark, on the which we might lovingly gaze,
   We have never beheld till now!

Lewis Carroll, *The Hunting of the Snark*

To those less fortunate
ACKNOWLEDGEMENTS

I wish to express my gratitude to Professor Mohamed Mansour for the opportunity to work at the Automatic Control Laboratory of the Swiss Federal Institute of Technology (ETH). In particular, I thank him for the freedom in which I could pursue my ideas. I sincerely thank Professor Walter Schaufelberger for agreeing to co-referee this thesis and for his helpful suggestions and questions. I am grateful to Dr. Essam Badreddin for his encouragement and support in various matters during the last years. His challenging inquiries and detailed feedback helped me a lot.

I extend my thanks to my colleagues (in alphabetical order) Dr. Remo Bless and Dr. Alois Holenstein, who beat me in the race for the completion of our projects. I shall miss their presence and cooperation. I also thank the other colleagues and staff members who made it a delight to be and work at the Institute. I regret that I cannot thank my late colleague Walter Grünenfelder for the times I enjoyed his clarity and wit.

I am grateful to my parents and my brother for their unobtrusive but steadfast support of my goals. I thank my dear wife for her patience and humour in putting up with me. Finally, I thank my Lord and God for his continuous and personal commitment to the humanity project.

Markus A. Müller
Zürich, January 1993
CONTENTS

Acknowledgements i

Contents iii

Summary vii

Zusammenfassung ix

Notations xi

1 Introduction 1
  1.1 The problem 2
  1.2 Example scenario 3
  1.3 Goal of the thesis 4
  1.4 Overview of the thesis 6
    1.4.1 Location 6
    1.4.2 Contributions 7
    1.4.3 Organization 7
  1.5 Frequently used terms 8

2 Issues in robot programming 11
  2.1 Robot programming 11
    2.1.1 Task specification 13
    2.1.2 Overview of planning functions 14
    2.1.3 Limitations of model-based planning 16
  2.2 Workspace Modeling 17
    2.2.1 Uncertain geometric data 18
  2.3 Sensing 19
    2.3.1 From sensor data to features 20
    2.3.2 From features to predicates 22
    2.3.3 Sensor planning 23
  2.4 Action planning 24
    2.4.1 Local versus global planning 24
    2.4.2 Gross motion planning 26
    2.4.3 Grasp planning 26
    2.4.4 Fine motion planning 26
    2.4.5 Operations planning 29
### 2.5 Related work

### 3 Uncertain geometric relations

#### 3.1 Uncertain Transforms

- 3.1.1 Representing uncertainty
- 3.1.2 Transforming uncertainty
- 3.1.3 Propagating uncertainty: Compounding
- 3.1.4 Integrating uncertainty: Merging
- 3.1.5 Interpretation of covariance matrices

#### 3.2 Comparison of uncertainties

- 3.2.1 Verifying the sufficiency of given information
- 3.2.2 Determining what information should be acquired

#### 3.3 Networks of uncertain relations

- 3.3.1 Requirements of assembly systems modeling
- 3.3.2 The network approach
- 3.3.3 Consistent integration of observations
- 3.3.4 Determining relative uncertainty

#### 3.4 Realization of network functions

- 3.4.1 Network reduction
- 3.4.2 Computing relative uncertainties
- 3.4.3 Observation integration

### 4 Modeling and planning

#### 4.1 Operation models

- 4.1.1 Mating: Joint kinematics
- 4.1.2 Mating: Pose update
- 4.1.3 Mating: Uncertainty update:
- 4.1.4 Unmating
- 4.1.5 Moving
- 4.1.6 Example of manipulation
- 4.1.7 Taking observations

#### 4.2 Knowledge model

- 4.2.1 Knowledge model definition
- 4.2.2 Class hierarchies
- 4.2.3 Update and retrieval operations

#### 4.3 Workspace model

- 4.3.1 Representation hierarchy
- 4.3.2 Representation of parts
- 4.3.3 Representation of relations

#### 4.4 Planning

- 4.4.1 The agent concept
- 4.4.2 Agent description
- 4.4.3 Planning the use of acting agents
- 4.4.4 Plan representation
- 4.4.5 Choosing sensors and features for sensing
5 Design and implementation
  5.1 System design .......................................................... 97
    5.1.1 Network functions and operations modeling .............. 97
    5.1.2 Workspace model ................................................. 98
    5.1.3 Planner and object level interface ........................ 102
    5.1.4 Robot interface and simulation .............................. 103
    5.1.5 Utility functions .............................................. 105
  5.2 Implementation environment ......................................... 105
    5.2.1 Hardware and low level software ............................ 105
    5.2.2 Software .......................................................... 108
    5.2.3 Prolog tools .................................................... 109
  5.3 Limitations of the current implementation ....................... 111

6 Examples
  6.1 Object and workspace modeling .................................... 113
  6.2 Uncertainty computation ............................................ 119
  6.3 Planning and operations modeling ............................... 120
  6.4 Operations modeling: Pose update ............................... 126
  6.5 Automatic sensing .................................................. 127
  6.6 Observation integration ............................................ 129

7 Conclusions and further work
  7.1 Conclusions .......................................................... 131
  7.2 Further work ........................................................ 131
    7.2.1 Uncertainty manipulation ................................... 132
    7.2.2 Plan fixing .................................................... 132
    7.2.3 Global error recovery ....................................... 133
    7.2.4 Versatile agents .............................................. 133
    7.2.5 Sensor planning ............................................... 134
    7.2.6 Data acquisition and learning .............................. 134
    7.2.7 Considering the broader context ......................... 134

A Knowledge model syntax .............................................. 137

Bibliography ............................................................. 139

Curriculum Vitae .......................................................... 145
SUMMARY

When programming an assembly robot, knowledge about geometric and logical object relations, and about the robot system's ability to sense and act is processed. In particular, not only geometric relations, but also their uncertainties are important. This is because sensors provide uncertain data and because the outcome of actions is uncertain. The resulting uncertainties about object positions (geometric uncertainties) must be bounded, or else the programmed robot actions may fail.

In industrial applications, uncertainties are limited by careful engineering of parts and handling devices, and by explicitly programming the use of sensors. A robot program incorporates the programmer's experience and reasoning with uncertainties. An attempt to automate robot programming and to exploit the flexibility offered by a robot system must also take uncertainties into account.

Knowledge about geometric uncertainties should allow a robot planner to maximize the success probability of actions and the information gained by sensing operations. For this, it is necessary to represent geometric uncertainties, model the effects of acting and sensing operations on uncertainties, and interpret uncertainties in a way suited to planning. Most existing approaches to handle geometric uncertainties are limited to either motion planning or to sensor data fusion.

In this thesis, a common framework for the treatment of uncertainties in both action and sensor planning is established. The uncertainties in the relations between objects in a robot's workspace are represented by a probabilistic model adapted from Durrant-Whyte. The relations may form an arbitrary network which must be consistently updated to reflect the robot's operations.

The main contributions of the thesis are methods for

- computing relative uncertainties in arbitrary networks of uncertain relations, based on an analogy to mechanical networks of compliant couples,
- modeling the effects of actions on uncertainties,
- visualizing and comparing uncertainties, and
- the integration of the above methods with a workspace model and an object level planner.

The workspace model is implemented in terms of a knowledge model incorporating class, typing and consistency maintenance concepts. A hierarchical planner uses the
workspace model to generate plans for pick and place operations and to choose sensors and features that provide required information.

Most of the resulting planning and modeling system has been implemented in the Prolog language. The function of the key methods and components has been demonstrated with a real robot system and by simulation.
ZUSAMMENFASSUNG

Beim Programmieren eines Montageroboters müssen die geometrischen und logischen Relationen der montierten Teile sowie die Fähigkeiten des Robotersystems in Betracht gezogen werden. Insbesondere müssen nicht nur geometrische Relationen, sondern auch deren Unsicherheiten verarbeitet werden, da sowohl Sensoren wie auch Aktuatoren fehlerbehaftet sind. Wenn die daraus resultierenden Unsicherheiten der Teilpositionen (geometrische Unsicherheiten) nicht begrenzt sind, kann das Programm versagen.


Die wesentlichen Beiträge dieser Arbeit sind Methoden

- zur Berechnung der relativen Unsicherheiten zwischen Knoten in einem beliebigen Netz unsicherer Relationen, basierend auf einer Analogie zu mechanischen Netzwerken aus nachgiebigen Elementen.
- zur Modellierung der Auswirkungen des Einsatzes von Aktuatoren auf Unsicherheiten.
• zur Visualisierung und zum Vergleich von Unsicherheiten.
• zur Integration der obigen Methoden mit einem Modell des Arbeitsbereichs eines Roboters und mit einem Objektniveau-Planer.


NOTATIONS

\( \mathcal{M}_{m,n} \) m-by-n real matrices
\( \mathcal{M}_n \) n-by-n real matrices
\( A, B, C, \ldots \) matrices
\( x, y, z, \ldots \) column vectors
\( I \) identity matrix in \( \mathcal{M}_n \)
\( A^T \) transpose of \( A \)
\( ^jT_i \) homogeneous transform matrix relating frames \( i \) and \( j \)
\( ^jD_i \) description vector of \( ^jT_i \)
\( k\Delta \) differential translation and rotation transformation, represented in frame \( k \)
\( ^kd_i \) differential motion vector, in frame \( k \)
\( ^kJ_i \) Jacobian of \( ^jT_i \)
\( ^k\Lambda_i \) uncertainty associated with \( ^jT_i \), in frame \( k \)
\( ^k\Upsilon_i \) information; \( ^k\Upsilon_i = ^k\Lambda_i^{-1} \)
\( ^k\Gamma_i \) maximum uncertainty of a physical joint
\( ^k\Sigma_i \) joint constraint matrix
\( ^jR_i \) relation; \( ^jR_i = (^jT_i, ^j\Lambda_i, ^j\Gamma_i) \)
\( \Phi \) indefinite information matrix of a network of relations

Syntax diagrams

'class', ',', '.'... terminal symbols
Rule-List, type... nonterminal symbols
Name, Class... identifiers

Prolog conventions

+, - input and output arguments (in procedure descriptions)
% comment; the rest of the line is ignored
Bla, Gru variables; begin with an uppercase letter
bla, 1.2 constants; strings beginning with a lowercase letter or numbers
[A,B,C] list of three elements
Chapter 1

INTRODUCTION

As one gets involved in robotics, one learns to admire the cognitive, reasoning and manipulative abilities of humans. The effortlessness with which we move around and handle objects lets the casual observer underestimate the complexity of everyday tasks. The continuing efforts to automate them have led to the recognition of their complexity, and have resulted in a great variety of mechatronic systems called "robots".

Different meanings are attached to the term "robot": While popular fiction usually depicts a robot as a machine with humanoid appearance, or at least intelligence, the Robot Institute of America uses a more pragmatic definition [LGF83]:

... a reprogrammable multi-functional manipulator designed to move material, parts, tools, or specialized devices, through variable programmed motions for the performance of a variety of tasks.

This definition asserts the manipulative abilities and the programmability of a robot. Another view is taken by [Nit85]:

A robot is a general-purpose machine system that, like a human, can perform a variety of different tasks under conditions that may not be known a priori

Here, the term "robot" is synonymous with "robot system", i.e. a combination of actuators, sensors, computers and auxiliary devices. It is stated that a robot should be not only "flexible", but also "intelligent", defining these properties as

"Flexibility" means the ability to perform a class of different tasks; "artificial intelligence" means the ability of a machine system to perceive conditions that may not have been known a priori, decide what actions should then be performed, and plan these actions accordingly.

Depending on the point of view taken, an autonomous land vehicle or a pick-and-place manipulator used for hard automation may be viewed as robots or not. Regardless of such labeling ambiguities, "robots" are being developed for applications such as vacuum cleaning, assisting impaired persons, gathering information and operating in hazardous environments.
In this thesis, we shall restrict ourselves to industrial robots used for part handling and assembly. Such a manipulator typically has four to seven serial or parallel links and is fitted with an end effector such as a gripper, screwdriver or glue applicator.

1.1 THE PROBLEM

In spite of their mechanical degrees of freedom and their programmability, manipulators still are extremely limited in their applicability to unconstrained real world problems. This shortcoming is due to

- the tremendous amount of real world knowledge and reasoning needed for planning even a simple task such as picking up and mating two fitting objects.
- the small number of sensors available to robots, the limited ability to interpret sensor data, and the insufficient integration of sensors, actuators and control strategies.

This is illustrated by comparing a robot system to the human hand-eye system. The human arm and hand are equipped, among others, with a great number of position, touch and force sensors, which also allow to judge an object's surface texture and to detect slipping. The poor positioning accuracy (for untrained persons) of the arm itself is improved by a stereo vision system which gives feedback on the relative positions of the arm and other objects. The most astonishing and least understood feature of the hand-eye system is its sensor data processing, multi level control and reasoning ability. While the technical sciences have developed a set of sensing and acting devices and combined them to fulfill some more or less useful tasks, we still are nowhere near integrated systems that make optimal use of all the available sensor information to control their actuators.

Humans seem to use such an integrated system: The observations made about the state of our surroundings are incomplete, inaccurate or erroneous, and the outcome of our actions is uncertain. Nevertheless, we are able to maintain a model of the world and to plan actions. The model must not be metrically exact, we instead rely on feedback when acting. When e.g. grasping an object, we observe and control the relative position of our hand with respect to the object while approaching it, without knowing where exactly the object lies. We do not want this knowledge per se, we are only interested in grasping the object.

In robotics, the development of sensing and reasoning capabilities needed to cope with an uncertain world still has a long way to go. Meanwhile, another approach has been taken in industrial automation: Metrically precise models are required, and any uncertainties are restricted as far as possible. Robots move to fixed, predetermined positions, and custom made fixtures and feeders ensure that the right part is at the right place at the right time. Remaining uncertainties are reduced by dedicated tools or by sensing, requiring careful engineering to determine where and how to use which sensor. The resulting robot applications impose many constraints on their environment. The constraints may be relaxed if planning, sensing and control capabilities are enhanced.
1.2. Example scenario

Whether this is economical depends on the application: In large batch production, cycle times strongly influence production costs. This warrants a large offline effort to reduce cycle times by optimizing operation sequences and using customized hard- and software. In the tightly constrained online environment, the time-consuming use of sensors is minimized. In small batch production, the relative costs for programming operations and designing the workspace increase. The automation of these tasks requires versatile fixtures, actuator and sensor hardware, together with flexible planning and control strategies. If the uncertainties are not eliminated by constraining the environment, they must be accounted for and eliminated actively by the robot system. For this, the system must be able to model the uncertainties, and contain strategies for coping with them.

Uncertainties can be classified and treated as follows:

1. Uncertainty in the order of part arrival and identity: Parts may either be held available in pallets or feeders, or arrive on a conveyor belt in a random sequence. Such uncertainties can be handled by online scheduling of assembly operations, based on information about their ordering constraints. This is the topic of a related thesis [Ble92].

2. Uncertainty due to failures and disturbances: Operations may have unexpected, usually undesired results. These may be due to position errors or the failure of a robot system component.

These uncertainties can, in principle, be handled by analyzing errors and re-planning operations, using symbolic, knowledge based planners. This is a very difficult task, for which no general solutions exist.

3. Uncertainty in part location and orientation, also called geometric or spatial uncertainty: Parts may be in a completely known position, or lying in one of several stable positions on a plane, or lie in a pile among other parts.

On the sensor level, statistical methods to filter sensor data are well established. The combination of observations from different sensors is the subject of current research. Still lacking are methods to model and reason about geometric uncertainties, taking into account the effects of acting and sensing. This is the topic of this thesis.

1.2 EXAMPLE SCENARIO

To give an impression of the kind of reasoning to be automated when considering geometric uncertainties, we informally describe a typical pick and place operation. Consider the situation in Fig. 1.1: The small block must be placed in the rectangular hole of the large block. As we have learnt in the first year of our life, this can be accomplished e.g. by picking up the small block, moving it to a suitable position above the large block, inserting it into the hole, and letting it go. The planner program also arrives at this sequence of operations: It ascertains that the gripper is not holding another part and that the parts can be grasped and mated. It determines a sequence
Chapter 1. Introduction

Figure 1.1: Initial workspace state. Two blocks and a pair of gripper fingers are shown. The uncertainty of the block positions with respect to some common reference frame is indicated by the 3D crosshairs. Linear uncertainties are denoted by plain lines, angular uncertainties by shaded lines. The directions of the lines give the axes of translation or rotation, and their lengths indicate the magnitude of uncertainty. The parts are known to lie on a plane surface, therefore their linear uncertainty is restricted to the horizontal (xy) plane, and their angular uncertainty to the vertical (z) axis.

of manipulator actions and their numerical parameters. When planning or executing these actions, the uncertainties and positions of the parts are updated continuously and evaluated when necessary.

Fig. 1.2 shows the workspace state prior to mating. If the relative uncertainty between the two blocks is too large, an uncertainty reducing operation may be introduced e.g. prior to grasping the small block (Fig. 1.3). If this is done, the new state prior to mating will be as in 1.4.

1.3 GOAL OF THE THESIS

The goal of this thesis is to provide a theoretical and practical framework for the treatment of uncertain geometric data in the planning and execution of robot operations, e.g. in

- choosing motion strategies for executing an operation by considering the location uncertainties of the objects involved and the accuracy requirements of the operation.
- evaluating the location accuracy requirements of an operation and deciding whether it has a sufficient chance of success, or whether additional location information is needed.
- choosing sensing strategies for acquiring object location information needed by an operation.
- the consistent integration of sensor data into a workspace model.

The goal is reached by modeling the relations between objects by a network of transforms with associated uncertainties. It requires, among others, the design and implementation of methods for handling geometric uncertainties by
1.3. Goal of the thesis

Figure 1.2: State prior to mating. The grasp operation constrains the position of the small block with respect to the gripper. Therefore the linear and angular uncertainties of the small block have been partly reduced. The relative uncertainty between the two blocks, shown to the left, is the sum of their individual uncertainties. If it is too large, it may not be possible to mate the parts. In the x direction, a larger error will be tolerated than in the y direction.

Figure 1.3: Uncertainty reduction prior to grasping. If the relative uncertainty cannot be absorbed by the mating operation, it must be reduced earlier: The uncertainty of the small block in the y direction may be reduced by grasping it in the y direction (left). If this is not possible, measuring the position of the circled edges may provide the necessary information (right).

Figure 1.4: New state prior to mating. Assuming that the observation shown in the last figure was made, the uncertainty of the small part and the relative uncertainty have been reduced as shown. If the relative uncertainty still is too large, the uncertainty of the large block’s position must be reduced. If it is small enough, the parts can be mated. The uncertainty of the large block after mating is shown to the right. It has been reduced because of the constraints enforced by the new joint.
Representing uncertain geometric relations between objects.

Interpreting uncertainties: It is required that the relative uncertainty between objects in the network can be computed and then visualized for human interpretation and compared for assessing operation feasibility or for determining in which direction uncertainties must be reduced.

Updating the network of uncertain transforms: Observations must be integrated consistently with existing estimates of the workspace state, and the effects of actions (i.e. when handling parts) on uncertainties must be modeled.

These methods must be combined with a system for modeling the workspace and any data needed for planning sensing and acting operations.

1.4 OVERVIEW OF THE THESIS

1.4.1 Location

Models of geometric uncertainty are required in different tasks, notably to evaluate robot errors, to fuse observations or to plan actions and observations. In a complete robot programming system, these capabilities should be combined and based on a common uncertainty model. Currently, many approaches are restricted to one of the above tasks. See [HS90] for a survey of observation fusion, and the following papers for some previous efforts related to sensing or planning. They are grouped according to their method of uncertainty representation.

Probabilistic uncertainty representation: [DW88] models arbitrary networks of uncertain relations and shows how to integrate observations consistently. An adapted version of this uncertainty model and integration procedure is incorporated in this thesis. [SC86] shows how to compute relative uncertainties, but is not applicable to arbitrary networks of relations. [MA89] propagate uncertainties through robot programs. Similar simplifying assumptions about motion under physical contact are made as in this thesis. Sensing is not considered, and the workspace is modeled by a tree of relations. [SL91] considers both the integration of observations and the modeling of the effects of actions, but does not discuss arbitrary networks of relations.

Set-oriented uncertainty representation: [PTP88] model the effects of actions on trees of uncertain relations. This allows to verify action plans by forward propagation of effects and backward propagation of uncertainties. Sensing is not considered. [HK90a] present a planner that considers uncertainty preconditions of operations. Symbolic expressions of uncertainty constraints are propagated backwards through actions to generate further constraints, which may be used to plan uncertainty-reducing operations.

The above approaches are limited to either acting or sensing alone, or allow only trees of relations, which does not adequately represent the real world. While [HK90a] makes
a more complete impression, we wish to replace its complex constraint manipulation system by simpler procedures for the forward propagation of uncertainties. They must be combined with heuristic procedures for fixing a plan when the uncertainty preconditions of an operation are not satisfied.

1.4.2 Contributions

The main contributions of the thesis are methods for

- computing relative uncertainties in arbitrary networks of uncertain relations, based on an analogy to mechanical networks of compliant couples,
- modeling the effects of actions, considering kinematic loops and loops of observations,
- visualizing and comparing uncertainties represented by $6 \times 6$ covariance matrices of linear and angular uncertainty, and
- the integration of the above methods with a workspace model and an object level planner that is designed to take uncertainties into account for both action and observation planning.

The implemented hard-and software system, named ARGUS\(^1\) works together with the M-EA\(\text{NS}\) system, described in a companion thesis [Ble92]. M-EA\(\text{NS}\) handles uncertainty with respect to the order of part arrival: It accepts orders for entire assemblies and then schedules the assembly of single parts, using the available parts and considering constraints on the order of assembly operations. The assembly commands, stated in terms of objects, are passed to ARGUS, which plans and executes robot level commands in a real robot workcell. This online version of ARGUS has only a qualitative model of uncertainties and uses fixed sensing strategies. A (currently) separate version of ARGUS incorporates a probabilistic uncertainty representation but is not connected to the real robot.

1.4.3 Organization

Chapter 2 introduces the issues encountered in robot programming and how they are approached in this work.

Chapter 3 develops the mathematical procedures used to represent and manipulate uncertain geometric information. Uncertainties can be combined, compared and visualized. The analogy between uncertain transforms and compliant couples is derived and used to compute the relative uncertainty between nodes of an arbitrary network. The implementation of the required computations, essentially a sparse matrix problem, uses a combination of graph manipulation and matrix methods.

\(^1\)Automatic Robot programming in the presence of Geometric Uncertainties, after the mythological hundred-eyed guardian appointed by the goddess Hera to protect the beautiful Io against the advances of Zeus. Argus must have had a formidable sensor data integration problem.
Chapter 4 incorporates the methods from chapter 3 in a workspace model. First, it is shown how positions and uncertainties are changed when handling and observing parts. The required object descriptions, together with data needed by the planner, are represented by a workspace model. In order to enforce some consistency, the workspace model is implemented in terms of a "knowledge model" which defines the structure of a database. A planner, given a goal, uses the workspace state and a standard description of sensing and acting capabilities to plan actions that achieve the goal. It models the changes of uncertainties, and plans the application of sensors to reduce uncertainties.

Chapter 5 describes the hardware and software structure of the implementation, user and data interfaces, tools of general use, and limitations of the current implementation. Chapter 6 gives examples demonstrating the application of the algorithms and programs presented in the preceding chapters. Chapter 7 contains the conclusions and a list of further work.

1.5 FREQUENTLY USED TERMS

This section introduces frequently used terms, which may later be explained in more detail. The scenario considered is a single assembly cell, incorporating one ore more manipulators with changeable grippers and a number of feeders and sensors. The terms manipulator and robot are used interchangeably. The robot system comprises the manipulator, sensors, other actuators and the computers controlling and linking them. The volume that can be reached by a manipulator is called its workspace.

From the point of view of sensing, the part of the workspace that can be observed is called the scene. The application of a sensor to obtain scene data is called an observation.

The complete computer representation of the workspace and the manipulator is the workspace model or just the model, while the part of the model representing the geometry of objects and their relations is the geometric object model, geometric workspace model or simply the geometric model. Symbolic or logic relations between entities, such as "A is part of B", or "C is affixed to D" are part of the logic model.

An object has six degrees of freedom, three linear, its position (or displacement) and three angular, its orientation. Position and orientation together are referred to as the object pose. The pose of objects is described by attaching (coordinate) frames to each object and by specifying the transformations between these frames. A special reference frame may be called the world frame. A transformation is described by a homogeneous transform matrix or its description vector containing position \((x, y, z)\) coordinates and orientation \((\phi_x, \phi_y, \phi_z)\) angles. The elements of the description vector represent a set of generalized coordinates. When referring to the direction of a motion, it shall denote a displacement in generalized coordinates, e.g. in the direction of the \(x\) or \(\phi_z\) axis.

The accuracy of a transform is described in terms of its uncertainty and expressed numerically by a covariance matrix, called the uncertainty (matrix) of the trans-
The term uncertainty, when used without further specification, refers to the uncertainty of transforms or geometric data in general. It may comprise linear, i.e. translational, or angular, i.e. rotational uncertainty. A transform with an associated uncertainty is called an uncertain transform (UT).

The task to be achieved by the robot can be specified on different levels of abstraction. They are the task level, object level and robot level. Standard object level actions are e.g. grasping, moving, placing or mating parts, and acquiring sensor information. They shall be referred to as operations, in agreement with the operator paradigm used in planning.

An operation is realized by an agent. An agent is an entity based on hardware and software that implements an operation autonomously. The description of agents is standardized and independent of their implementation.

Planning is done on two levels: Operations planning assumes the existence of agents with specified abilities and plans the application of these agents with no knowledge of their interior working. Sensor planning, grasp planning and so forth plan the execution of a specific agent’s operation.
ISSUES IN ROBOT PROGRAMMING

This chapter shows the issues involved in the realization of an object level planning system. Existing approaches for handling these issues and their treatment in the ARGUS system are presented. The first section introduces the main problems which must be addressed when programming a robot. The following sections then show the issues related to modeling, sensing and action planning in more detail. Finally, a number of task level robot programming systems and systems incorporating the modeling of uncertainties are described.

While written text displays information sequentially, along a single dimension, the real world and its representation within the human brain are anything but one-dimensional. The graph in Fig. 2.1 shows some of the issues encountered in this thesis.

2.1 ROBOT PROGRAMMING

The motivation for the research into higher level robot programming techniques is stated in the classic paper [LP83]:

The key characteristic of robots is versatility; they can be applied to a large variety of tasks without significant redesign. This versatility derives from the generality of the robot’s physical structure and control, but it can be exploited only if the robot can be programmed easily. In some cases, the lack of adequate programming tools can make some tasks impossible to perform. In other cases, the cost of programming may be a significant fraction of the total cost of an application. For these reasons, robot programming systems play a crucial rule in robot development.

The various robot programming levels already existing or under development are introduced in practically any book or article on robotics, see e.g. [LGF83, Pug83, LP83, Nit85]. They shall therefore only be listed briefly here, in ascending order of abstraction:

Teaching or guiding, specifying robot positions and motion trajectories by moving the robot itself and by recording position encoder readings.
Figure 2.1: Issues and relations. The edges of the graph are undirected and represent relations of the type has_something_to_do_with. The thicknesses of boundaries indicate the weights of issues within the thesis.
2.1. Robot programming

Robot level programming, specifying actions in terms of robot positions. Sensing requests return sensor specific data.

Object level programming, specifying actions in terms of the objects being handled.

Task level programming, specifying actions in terms of their final result (or goal state, using planning terminology). The term "task level" is not used consistently in the literature. It sometimes is used for the lower level, which here (and e.g. in [ACC87]) is called the "object level" or "feature level".

The task and object level specifications are meant to be independent of robot and sensor characteristics. Since the corresponding programming systems require a model of the robot and its environment, they are also called world modeling systems or model based systems.

2.1.1 Task specification

Whatever the level on which an assembly process is specified, the goal relations between objects are always involved, be it implicitly or explicitly. While the possibilities for specifying actions at the robot level are limited, there are different approaches for object and task level specifications. They can be stated by a human programmer or be derived from CAD models of the finished product, and they can be expressed in terms of the relative pose of objects or in terms of spatial relations between object features. The approaches based on CAD models often use a solid modeling system to determine how the finished product could be taken apart. This gives the geometric relations between objects and the motion trajectories for mating them. The disassembling sequences can be used to generate constraints on assembly precedences [Blec92].

The representation of the relative pose of objects using cartesian frames can be extracted from CAD models in a straightforward manner, but its use has some drawbacks (partly from [LLLM84]):

- Frames are difficult for humans to reason about. This makes it necessary to have a graphic system for specifying and visualizing object positions.

- A frame specifies poses completely. This means that they are overspecified when relating symmetric objects or features. Partial constraints may be expressed by flagging the corresponding symmetry axes.

- Frames are only appropriate for modeling rigid objects. This is also the case of most available geometric modeling techniques.

- Frames contain no information about the geometry of the joints between the parts, which might be needed to derive fine motion strategies.

The first and last points stated above advocate the use of spatial relations between object features for specifying object relations. The earliest system of this kind, RAPT [Pug83], uses object models and a description of the goal relations between surface
planes and cylinders to derive constraints on the relative pose of the objects. Similar systems are AUTOPASS [LGFS83], LM-GEO [LLLM84] and TWAIN [LPB86], from which this example of a goal specification is taken:

\[
\text{PLACE A SUCH THAT} \\
(A.4\ \text{AGAINST TABLE}) \text{ AND} \\
(A.1\ \text{AGAINST F.1}) \text{ AND} \\
(A.2\ \text{AGAINST F.2})
\]

where A and F are objects, and where the suffixes identify their surfaces. The task planner uses this specification to plan compliant motion and sensing commands.

The task specifications used by ARGUS are processed by the M-EANS system [Ble92]. M-EANS uses a geometric model of the finished product, together with manually generated assembly trajectories, to generate precedence constraints on assembly operations offline. An online scheduler takes the constraints into account and generates object level commands that are implemented by ARGUS.

ARGUS specifies object relations by frames. The drawbacks mentioned above are partly overcome by

- procedures for displaying the structure and the numeric values of object relations, and a 3D wireframe display of the workspace model state.
- representing partial constraints between objects by associating uncertainties with their relative transform.
- allowing to represent the geometry of joints in terms of standard joint types and their tolerances. This should allow fine motion strategies to adapt to the current mating task.

2.1.2 Overview of planning functions

Regardless of the level of programming and of the application, the following problems must be addressed by either the programmer or by a machine, assuming that a robot system with known sensory, manipulative and computing capabilities is available (partly from [LPB83]).

Workspace layout: The type and the positions of feeders and fixtures must be determined. Feeders provide the parts being assembled, fixtures hold the parts during assembly or to store them temporarily. They may be chamfered to reduce pose uncertainties passively. Their locations must be chosen to minimize cycle times. Fixtures may be designed automatically, using standard reusable elements.

Scheduling of operations: The sequence of operations (see the following items) must be determined. Depending on the type of application, this will happen either offline or online. In a tightly constrained environment, where the arrival time of parts is known, the sequence of operations may be chosen in advance. With parts arriving randomly, it must be determined online, taking into account assembly precedence constraints and the costs for buffering and changing grippers.
Grasp Planning: A relative position between gripper and grasped object must be determined that ensures a stable grasp and avoids collisions with other parts or fixtures during grasping and mating.

Gross Motion Planning: An efficient path for positioning the manipulator and the grasped objects must be determined, avoiding collisions.

Fine Motion Planning: Strategies for sensor-controlled motion must be determined, that result in the reliable execution of grasping and mating operations in spite of position uncertainties.

Sensor Planning: The above operations all use sensors, be it to acquire information prior to planning, or to verify the correct execution of an action. In both cases, a sensor-feature pair that provides the required information must be determined.

These planning problems all are related to the issue of modeling the robot’s environment and actions. Models may be explicit or implicit. Explicit modeling describes the world independent of actions. In artificial intelligence (AI) terminology, this approach is called the interpreted-symbolic-structures approach. It is based on a fundamental methodological assumption of AI, the knowledge representation hypothesis, which may be summarized as [Seg88]:

any intelligent system will require knowledge about its domain that is explicitly and recognizably encoded.

Taken to the extreme, this leads to a system structure represented by the loop shown in Fig. 2.2.

On the other hand, it is possible to describe actions directly in terms of sensor data. This reflexive behavior or situated-automata approach uses an implicit world model. Different levels and strategies of behavior may interact and possibly conflict with each other.

We believe that no single paradigm or tool is suited for all problems. Assembly tasks require some minimal model of the parts and products being assembled, at least for describing how parts are put together or what the finished product ought to look like.
As flexibility increases, more explicit models and more abstract task specifications are required.

Although the remainder of this chapter is structured according to the explicit modeling approach, the implementation of ARGUS contains a variety of loops of information flow (Fig. 2.3): On the upper level, operations planning is guided by qualitative information extracted from the workspace model. On the lower level, operations for manipulating and observing objects use geometric data retrieved from the workspace model, as well as sensor data in their local control loops. These lower level operations are well suited for the implementation of reflexive behavior.

### 2.1.3 Limitations of model-based planning

Robot and workspace models are commonly used to simulate the geometric aspects of robot actions: Simulations of robot kinematics and dynamics, including 3D visualization and collision detection, may verify planned trajectories. Simulation tools are commercially available, but their accuracy is limited by the accuracy of the identification of link lengths, compliance and servo loop parameters. For example, the calibrated positioning accuracy of the industrial SCARA robot (an IBM 7575) used by ARGUS is ±0.08mm, while its repeatability is ±0.025mm.\(^1\) The calibrated accuracy is achieved only within part of the workspace. Only two links and joint positions are involved, and the influence of gravity and payload is small due to the SCARA geometry. The

---

\(^1\)The positioning accuracy is the ability to reach a point specified in some world coordinate space. It involves the solution of the backward kinematics problem, which depends on robot link parameters, to transform the point into joint space. The repeatability is the accuracy in reaching a position specified in joint space, found by guiding. The resolution is the smallest displacement that a manipulator can be commanded to move [Wol87].
manipulator is relatively accurate, if compared with other arm geometries: For large, articulated robot arms with six or more degrees of freedom, the factor relating positioning accuracy to repeatability increases, which is why robot manufacturers usually state only the repeatability in their product descriptions. Further uncertainties lie in the location and dimensions of fixtures and parts, and in the effects of actuators on the workspace.²

Because of errors in modeling these geometric relations, it may still be necessary to teach manipulator positions in the real workspace. This eliminates the influence of constant parameters such as manipulator link lengths and fixture positions. Time-varying errors such as errors in part dimensions and poses must still be eliminated online.

One approach for coping with pose uncertainties is to chamfer parts, grippers and fixtures, such that uncertainties are reduced passively when grasping or mating parts. But not every part may be chamfered and chamfering does not solve all problems. If a robot system must cope with uncertainties and be applicable to a variety of tasks, it needs external sensors (in addition to the internal position sensors of manipulators). They entail an engineering effort to integrate them into the robot system and to determine when and where to use them. This process is — whether consciously or not — based on reasoning about uncertainties and the effects of acting and sensing operations. In ARGUS, a basis for the automation of this reasoning process is created.

2.2 WORKSPACE MODELING

As concluded in the previous section, some degree of explicit modeling is needed for planning and for reacting to unexpected events. In robotics domain, this is reflected by the importance of a geometric workspace model, realized by a solid modeling system. However, the model is a means and not an end in itself. Its data is used by the other functions of the robot system, which may require different representations of geometric object data. A geometric model should be independent of the remaining system, and general enough to provide the information that the system needs. It must be complemented by information about uncertainties and symbolic relations, and provide mechanisms for maintaining them. A complete model, used for task and object level planning, contains:

- a geometric model of individual objects that may turn up in the workspace, including the manipulator itself.
- other physical properties of these objects
- functions for generating operation-specific object data, such as grasp points, feature locations and object representations used by a vision system. These functions may also be associated with specific operations instead of being part of the common geometric model, and an agent may itself maintain object data derived from the common geometric model.

²A fundamental and readable discussion of the unpredictability of classical mechanics is [Pen89].
• geometric relations of objects. They specify the relative position of objects numerically and include the possible relations, which describe legal assemblies, and the actual relations, which describe the current state of the workspace.

• symbolic relations of objects, both possible and actual. Symbolic relations may be statements like "Object A is on object B", "Object A is being grasped" or "Surface 5 of object A fits against surface 3 of object B".

• functions for modeling the effects of operations and updating the geometric and symbolic relations. They may use either feedback on the outcome of an executed operation, or determine it by simulation.

• a representation of tolerances on the dimensions of objects and of uncertainties in geometric relations. The latter arise mainly because of sensor uncertainties. Modeling geometric errors explicitly allows to reason about the possibility of success of an operation, and about where and how to sense in order to acquire geometric information. Note that it is not necessary that the geometric model be arbitrarily perfect. The success of an operation is ensured by some limited accuracy of the model. Greater accuracy is of no use and may be a source of inefficiency, caused by superfluous sensing.

In ARGUS, mainly the last four points are addressed. Object geometry is modeled only as far as the relations between objects are concerned. In addition, each object is described as a combination of cubes and cylinders, allowing a graphic display of the workspace model state.

2.2.1 Uncertain geometric data

The uncertainty in object locations and in relations between object features must be represented and manipulated in a manner that is consistent with geometry and probability theory. Uncertainties can be determined from

• data on the parts being handled, i.e. their manufacturing tolerances.
• actuator models: Uncertainties in the positions of actuators are propagated to the positions of the manipulated objects. They are related to robot positioning accuracy and the accuracy of feeders and fixtures in which parts are introduced into the workspace.
• sensor models: Sensors accuracy is limited, and position sensors usually provide only partial information about the observed object.

There are two approaches for modeling geometric uncertainties: The set-oriented and the probabilistic approach. In the set-oriented approach, variables are associated with upper and lower bounds. When objects are manipulated, error bounds are propagated through operations, and their influence on the outcome of operations is evaluated. This can be done numerically [Tay76] or symbolically. Symbolic analysis may use the SUP/INF method [Bro82, HK90a]: The functions SUP and INF take a symbolic
2.3. Sensing

expression and a set of variables, and return upper and lower bounds on the expressions in terms of the variables in the variable set. This allows to reason backwards from uncertainty conditions to initial state parameters.

In the probabilistic approach [SC86, DW88, MA89, SL91] variables are considered to be (usually gaussian) random variables. Problems of interest are the modeling of the effects of operations on uncertainties and the maintenance of consistent networks of uncertain relations. A numeric representation of uncertainties allows to simulate acting and sensing operations straightforwardly. However, it is not possible to reason backwards: If e.g. the information about an object’s pose was collected by several sensors, it is afterwards not possible to determine the contribution of each sensor.

When reasoning about uncertainties, a fuzzy rule-based system might be used. A possible rule is

\[
\text{IF the gripper constrains the part pose in direction } x \\
\text{AND the uncertainty in direction } y \text{ is large} \\
\text{AND the required uncertainty in direction } y \text{ is small} \\
\text{THEN observe the part pose in direction } y \text{ prior to grasping.}
\]

Such rules would require a model that says whether uncertainties are large or small in specific directions. With only two or three degrees of freedom and rectangular objects, this might be accomplished by a collection of facts such as

"The relative uncertainty between the gripper and part A is small in the direction } x \text{ and medium in direction } y."

where the } x \text{ and } y \text{ direction vectors would have to be represented in a geometric model. With increasing generality and number of degrees of freedom, such a model will tend towards a complete workspace and uncertainty model.

ARGUS therefore uses a general, probabilistic model to simulate the effects of actions and observations. It can be used by a variety of planning techniques, including strategies incorporating knowledge as expressed by the example rule. Currently, planning knowledge is implemented by heuristic procedures.

2.3 SENSING

Sensors reduce the uncertainty about the state of the world. The uses of sensors can be classified as follows [LP83]:

- Initiating and terminating motions: waiting for feeding devices and guarded moves.
- Choosing among alternative actions: Testing for the success of a grasp or mate operation or inspecting an object.
- Obtaining the identity and position of objects and features of objects.
• Complying to external constraints: Moving continuously in response to continuous sensory input.

In a robot level system, the reaction to sensor readings is programmed explicitly. Typically, 80%-90% of program code is used to handle errors and sensors. In a model based system, sensor data (if it not used locally to control an operation agent) is interpreted in terms of the model, causing an update of the model, which then may influence further action planning. Issues in model-based programming and sensing are: extracting features of the raw sensor data, interpreting these features in terms of model features (matching), fusing information from different sensors and a priori information, planning the use of sensors, and predicting the results of observations. The naive order of application is planning, predicting, extracting, interpreting and fusing. To make an optimal use of information, these steps should not be traversed sequentially, but interact with one another:

• Prediction can guide interpretation and, on the lower level, feature extraction.
• The fusion of observations can cause interpretations to be reassessed.
• Planning depends on sensor capabilities and previous observations. It can be guided by prediction, i.e. by an analysis whether a given observation can be expected to provide the desired results.

The remainder of this section introduces different sensors and their applications, the interpretation and fusion of feature data, and the planning of observations.

2.3.1 From sensor data to features

A large variety of sensors and related sensor data processing techniques are used in robotics. Some (mainly image analysis) share applications with other fields of research. Sensors may be classified as contact or noncontact, and may provide either sparse data or arrays of data points (images). Most sensors providing geometric information use visible or infrared light, which makes them dependent on the optical properties of objects. Others, such as capacitive or inductive sensors, are material dependent.

The following list shows sensor types used in robotics, and the data they provide. It is followed by an overview of how they are used.

Vision, using video cameras provides 2D binary, grey level or color images. Cameras may be stationary or mounted on a manipulator. Depending on the image analysis capabilities, object and background color as well as lighting conditions have to be constrained.

Range imaging provides 2D range images, using optical triangulation for measuring distances. Triangulation is done actively by scanning with a light point or projecting light patterns, or passively, using stereo images or sequences of images. Range images are incomplete, as features may be occluded by shadowing.
2.3. Sensing

Optical reflectance sensors emit (usually infrared) light and measure the amount of light reflected. It is influenced by distance, surface roughness, color and reflectance. If surface properties are known, the distance can be estimated. Disturbances by other light sources are eliminated by using modulated light.

Ultrasonic distance sensors measure the time the echo of an ultrasonic sound burst takes to return. Their angular resolution is low, making them suited for collision avoidance in both manipulator arms and mobile robots.

Surface orientation sensors use an array of optical or inductive sensors. The combined sensor readings allow to estimate the orientation of a surface relative to the sensor array.

Touch (or haptic) sensors use various physical effects to detect contact or slip and to create tactile images.

Force sensors measure the deformation of a body, usually with strain gages, piezoelectric elements or optical triangulation, and compute the forces acting on the body. Several six-axis force-torque sensors are commercially available.

In certain situations sparse data is sufficient and can be used immediately for e.g. motion control or to verify predictions. Sensors providing sparse data can be used 1. for applications where little uncertainty is left: Grasping and tracking moving objects, 2. for fine motion operations: approaching and departing, moving without collision, orienting the gripper normal to a surface, scanning a surface and finding its axis of symmetry, and 3. for recognizing objects and their position. This last application motivates the automatic generation of optimal sensing strategies for identifying and locating objects.

Two-dimensional images, provided by vision, 3D-scanners or tactile arrays need more processing to extract image features such as points, lines and surface patches. With optical sensors, viewpoint dependent artifacts may be observed, e.g. an occluding edge seen when looking at a curved surface. Artifacts must be distinguished from actual surface boundaries in the interpretation phase. The analysis of range images uses approaches analogous to those of 2D image analysis, with different operators for image segmentation. A good collection of 3D perception techniques is [Ros86].

ARGUS assumes that a set of sensors for locating simple geometric features (vertices, edges and surfaces) is available.

If a 2D or 3D image sensor could provide any required information, then it would not be necessary to reason about which information might be needed and how to acquire it. The sensor would have to

- be able to identify and locate objects whose pose is unknown or partially constrained. Currently, no general solutions to this problem exist.
- be reliable and affordable.
- be able to observe the entire workspace.
- provide some minimum accuracy. For the sensors available today, this requirement conflicts with the preceding one.
A workspace model, not necessarily incorporating uncertainties, would still be needed for planning, and it would greatly facilitate the interpretation of scene data by providing predictions. Detailed information requests, as provided by ARGUS, would allow even more efficient data acquisition.

Because of the above requirements (mainly the first two), the use of 2D and 3D image sensors has been restricted to systems used for research and industrial applications under tightly controlled conditions. A realistic application may have an image sensor with limited accuracy and field of view, complemented by more accurate sensors providing sparse data. Such a setup can profit from modeling and planning with uncertainties, as shown in the ARGUS project.

2.3.2 From features to predicates

Interpretation

Quite a lot (!) of work is being done on image interpretation. Interpretation relates sensor features to model features. In model based vision systems, image points, lines and patches are interpreted in terms of modeled object vertices, edges and surfaces. The approach taken in commercially available vision systems uses vision specific features of parts lying in one or more of their stable positions. Features are extracted in a teaching phase, eliminating the need for explicit geometric modeling.

The naive approach to interpretation matches observed features to object features for each expected object. The best match then identifies the observed object. This approach fails due to the explosion of computational effort, if the variety of objects and their degrees of freedom is not very constrained. The computational effort may be reduced by using features invariant under rotation or general transformations (e.g. parallel lines), appropriate pruning of the search tree, or simulated annealing. Any a priori information about object pose and identity reduces the search space significantly, and if the information is complete, then the interpretation step is reduced to verification.

Fusion

Observations must agree with expectations and must be integrated into the current workspace model, maintaining the consistency of geometric relations. Integration interacts with interpretation in the following way: If an observed feature is matched to a model feature, the observation must be integrated with expected model data. The observation may

- verify the model, agreeing with expectations.
- complement the model, providing new information. In the case of pose information, observations are of limited accuracy or may constrain object poses only partially. Subsequent observations then may complement them by providing information that is more precise or that constrains positions further.
• contradict the model. Either the interpretation must be reconsidered, or the model must be revised, or the sensor must be assumed to be faulty.

For a survey of multi-sensor fusion, see [HS90]. ARGUS focuses on the fusion of observations and delegates the interpretation of image data to a separate vision system. The methods used to integrate geometric observation data are presented in chapter ??.

2.3.3 Sensor planning

The basic idea of automatic sensing is to use a priori knowledge and the expected behavior of objects and agents in order to guide the use of sensors. Planning the use of sensors is also related to action planning, notably to fine motion planning. Fine motion planning and compliant motion, which often are based on contact sensing, are discussed in section 2.4.4. This section considers noncontact sensing used to acquire pose information.

If a sensor makes an observation, it has already been decided that the observation is necessary at all. Therefore, generating perception requests is the first step of sensor planning. The next step is to decide which combination of features and sensors will provide the required data. This step, and the interpretation of the observation, are assisted by the prediction of observed features.

Generating perception requests How can the need for sensing be determined? Sensing reduces uncertainty about geometric and symbolic relations, i.e. about the position and existence of objects. A sensor is therefore used when some uncertainty is likely to cause the failure of an operation.

A human programmer knows when to anticipate such situations either by reasoning alone or by experimentation. An automated system may use the same approaches: It may model uncertainties and their propagation explicitly and determine whether an operation is likely to succeed. It may generate expected sensor values and insert sensing commands to check the outcome of operations. It may also monitor and analyze the execution of the operations, and, in the case of failure, propose sensing operations to be performed in future executions of the plan.

In ARGUS the maximum uncertainty that can be tolerated for specific operations is represented explicitly. During planning it is compared with the modeled uncertainties. If uncertainties are too large, sensing is guided by their difference from the given maximum uncertainty.

Selecting features and sensors Given a perception request, the selection of observations must take the following possibilities and constraints into account: Several sensors may take observations at several positions, providing complementing information. Sensors differ in their availability, accuracy and the capability to observe a given feature, and visibility and focus constraints reduce the space from which a given
feature can be observed. The selection of sensors and features should balance the information gained against the required sensing effort.

ARGUS concentrates on the information requirements, ignoring most of the other constraints.

**Predicting observations** The selection of features to provide needed information depends on a prediction of how a feature will appear to a given sensor. The expected scene data is mapped to sensor data, using models of sensor physics and geometry.

ARGUS does not generate predictions in terms of sensor data, but it uses the expected poses of features to predict the information they provide about their object. The verification of a prediction is achieved by the successful observation of the feature and its fusion with the workspace model.

### 2.4 ACTION PLANNING

The interacting control loops in a structure as in Fig. 2.3 are specified at various levels. At the hardware level, servo controllers control actuator positions. At an intermediate level, procedural programs link arbitrary sensors and actuators, implementing fixed sensing or acting strategies. For example, a manipulator is used to scan the workspace with a camera or a distance sensor, and a force sensor is used for mating two parts. At the highest level lies the planning of actions that achieve a specified goal, which continues to be a favorite of AI research. It may become arbitrarily complicated, depending on the degree in which constraints on parts, their positions, the workspace, its environment and the task at hand are relaxed.

The main operations in assembly are grasping, moving and placing or mating objects. Accordingly, planning has traditionally been divided into grasp, gross motion and fine motion planning, and their coordination, which we shall call operations planning. This division is misleading, since e.g. grasp planning depends on the situation when mating; at the same time, the motions executed when mating depend on how an object has been grasped (Fig. 2.4). In general, decisions made when planning one operation constrain the execution of the other operations. For the generation of a globally optimal plan (with respect to execution time and chances of success), a planner with knowledge about the domain of each single operation is required. Quoting [LPB83], we may say:

"These dependencies conspire to make robot operations appear monolithic; one often concludes that everything must be decided before anything at all is decided."

#### 2.4.1 Local versus global planning

These dependencies are taken into account in [VMT90] by first generating grasp points which then are evaluated by planning gross motion and mating operations for each
2.4. Action planning

![Image: Diagram of a gripper and two parts, Part A and Part B, with constraints indicated]

**Figure 2.4: Grasp point constraints.** The goal state shown, with part A placed in a corner of part B, constrains the grasping possibilities of A. If conflicting constraints exist when picking up A, it must be grasped with one grasp point, placed on the table and regrasped with another grasp point.

A grasp point. Such a global planner must work offline, considering all details of agent operations and the propagation of their effects. It outputs a complete robot level program which requires a tightly controlled environment for its execution. Constraints can be relaxed by generating grasp points and mating trajectories for all stable orientations in which parts may enter the assembly cell. Still, not all constraints imposed by the online situation can be eliminated, such as collisions or the reachability of a grasp point. As more flexibility is required, more online planning is required, making the time used for planning crucial for the efficiency of the application.

A common and powerful method for problem solving is the hierarchical decomposition into subproblems. If planning can be decomposed, then locally capable ("intelligent") agents may reduce the load on higher level planning, speeding up online planning. The implementation of individual agents can be optimized.

Problem decomposition is hindered by the dependencies between operations. They are due to the effects of geometry and errors, which are global. Geometric dependencies are created by the need for stable grasp points and collision free approach, transfer and depart trajectories. Error dependencies influence e.g. the choice of a grasp point and a grasp strategy: They are determined by the uncertainty in the relation between the gripper and the object to be grasped and by the error tolerances of the operation to be executed with the grasped object.

The dependencies that cannot be eliminated must be described and processed in a way that is independent of individual agents. Then constraints imposed by individual operations can be propagated forwards and backwards through a plan [LPB83]. In ARGUS, constraints are expressed in terms of geometric uncertainties of object relations.

We now discuss individual operations, followed by their integration through operations planning.
2.4.2 Gross motion planning

Gross motion planning determines a collision-free path for moving one or several objects to a goal position (usually the approach point for a grasping, mating or sensing operation). The difficulty of the problem increases with the number of degrees of freedom of the manipulator(s) and with the degree to which their environment is cluttered.

In a reasonably laid out workspace, and when handling objects that are small with respect to the workspace size, gross motion planning can be greatly simplified: The simplest approach, taken in ARGUS, is to define a volume of the workspace which always is empty except for the manipulator and from which the entire remaining volume containing parts, feeders and fixtures can be reached without collision.

2.4.3 Grasp planning

Grasp planning determines stable grasp configurations for a given gripper and object. They allow to grasp and handle the object without slipping. Grasp planning interacts with other operations by their constraints on the grasp configurations. It may be necessary to place and regrasp an object in order to orient it properly for a mating operation [HK90a, HK90b], or to use a second manipulator to which the part is passed [Hö92].

A complete analysis of these constraints runs contrary to the goal of distributing and decoupling planning. A robust grasp procedure should adapt to a cluttered environment and compensate for pose errors. A system for grasping known objects in an unknown environment has been implemented by [LIT90]: An object model is used to find opposing parallel faces of an object, by which the object may be grasped. When executing the grasp operation, a vision system and 3D range sensor observe the object and its environment. From this data, the final grasp point and a safe trajectory for reaching it are determined.

ARGUS uses grasp points provided by the M-EANS system, which checks the interference of grippers and parts during grasping and mating. Online "intelligence" is limited to collision detection.

2.4.4 Fine motion planning

Fine motion planning determines a strategy for achieving a given relation between objects whose relative position is uncertain. It may use [LT86]

- Goal oriented motions to achieve required relations
- Information gathering motions to reduce position uncertainty
- Conditional statements, which implicitly interpret sensory data in terms of spatial relations requiring different courses of action
- Loops to repeat goal-oriented motions that were previously unsuccessful
2.4. Action planning

Figure 2.5: Fine motion strategy example. Consider a simple peg in hole task with uncertainty along one horizontal axis. The following strategy — not the best — is applicable: The peg is deliberately offset by a distance larger than the expected error, such that a search must be made in only one direction. The peg is moved downward until contact occurs (a). It then is moved in the direction of uncertainty, using force feedback to maintain a small downward force (b). If the peg slips into the hole (c), horizontal reaction forces increase. When the increase is detected, the motion is stopped. Now the peg must be inserted, avoiding wedging and jamming (d). Although simple, the task depends on a number of physical factors such as friction, stiction, surface roughness, exact contact geometry and properties of the control loop.

- Variables representing uncertainty, and assignments to update them after either corrective or information gathering motions have been executed

An important achievement would be to have a set of strategies for generic assembly operations. The ultimate goal is the automatic generation of motion strategies, given models of the parts being assembled, or given the parts themselves.

The main work in fine motion planning is done on mating operations, but uncertainties may also be reduced by grasping and pushing objects on a plane. This is possible, although the exact distribution of support forces of an object lying on a plane cannot be known [Mas85, Mas86].

Approaches to the fine motion planning problem have been based on procedure skeletons, learning, and geometric reasoning. In the procedure skeleton approach, a fine motion strategy is selected, and constraints on initial positions and motions are determined, based on the propagation of error bounds by numeric or symbolic methods [Tay76, Bro82]. The symbolic analysis of error bounds involves a large effort in symbolic computation. Many errors are difficult to estimate, and error propagation methods based on worst case assumptions give overly constrained results.

In an approach to automate the learning of motion strategies, [HD86] record the positions and forces during multiple guided executions of the same force-controlled mating operation and generate a motion strategy. When implementing a motion strategy, the actual forces will differ from the expected ones. Control parameters and thresholds should adapt to these changes.
[LT86] automatically generate sensor-based fine-motion movements using geometric reasoning and symbolic constraint manipulation. The different ways to disassemble an object are determined, and the moving constraints imposed by the object contacts are represented symbolically. From the disassembly graph, a sequence of guarded compliant moves is generated.

**Compliance**

Compliance is very important to fine motion planning if uncertainties exceed tolerances. The basic idea of compliance is [Mas82]:

> [...] Common to all these tasks is that the trajectory is modified by contact forces or tactile stimuli occurring during the motion. A motion is compliant to the extent that it departs from the rigid positioning paradigm which dominates robotic manipulation. Compliance occurs either because the control system is programmed to react to force or tactile stimuli, or because of passive compliance inherent in the manipulator linkage or in the servo. If we define compliance to mean the ability of a manipulator to react to contact forces or tactile stimuli as the motion proceeds, then we see that compliance lies at the heart of one of the key problems of robotic manipulation, which is to integrate sensing into a manipulator programming and control system.

Various motion strategies, compliance mechanisms and control strategies for active compliance have been developed and applied. If parts are chamfered, the remote center compliance (RCC) [Whi82] can passively compensate alignment errors greater than 1mm. If parts are not chamfered, similar errors can be compensated using active compliance.

A description of active compliance behavior can be given by a damping matrix (or its inverse, an accommodation matrix) which relates measured forces to corrective motions. For simple mating operations such as chamfered peg in hole, a center of compliance located at one point in space is sufficient to achieve the operation. In this case, the accommodation matrix can be diagonalized. In other cases, non-diagonal accommodation matrices are appropriate. [Pes89, SP90] show how to synthesize accommodation matrices for other operations, based on an analysis of possible erroneous contact configurations. [Str87] gives strategies for the generalized peg in hole task (the peg and hole having an arbitrary prismatic form), using sequences of moves described by diagonal accommodation matrices.

The practical realization of compliant operations has been restricted mainly to systems used in research. Commercially available robot systems, for security reasons, usually do not allow to integrate external sensors into their control loops.

In ARGUS, a force-torque sensor can be used to control robot motion, restricted to the four degrees of freedom of the available manipulator. Compliant behavior is specified by a diagonal accommodation matrix and terminating conditions on forces and positions in each of the four axes.
2.4.5 Operations planning

Planners applied to robot assembly planning can be classified as Domain independent and dedicated planners. Domain independent planners — the classical AI planners — concentrate on the logic of the planning process, given the logic relations and effects of a set of operations available to a system. They lack an understanding of geometric relations. Dedicated planners for robotic assembly tasks include the generation of operation sequences, the analysis of high level task descriptions, motion planners and error recovery planners.

In the ARGUS/M-EANS system, M-EANS does offline planning and online scheduling of object level operations [Ble92]. They are passed to ARGUS which then plans and executes robot level operations. ARGUS's planner is dedicated to the use of uncertainties and limited to the planning of single object level commands.

2.5 RELATED WORK

Research on robot programming made robot level languages such as VAL-II and AML/2 commercially available in the early eighties. In AI, domain independent planners were developed and applied to blocks world planning problems. In robotics, world modeling systems were developed as a basis for object and task level programming (RAPT, LM, SRL). Nowadays, ever more research is directed towards task level programming, domain specific planning based on geometric data and the automatic use of various sensors, the integration of sensors into the control loop, vision and range image systems, and the structure of robot operating systems.

RAPT (University of Edinburgh)

Work by Ambler, Popplestone et al. on the RAPT system began around 1975 [PAB78, ACC87, Pug83]. RAPT describes an assembly task by the spatial relationships of object features, similar to the example from TWAIN given earlier. An inference system determines relative positions of the objects that satisfy these spatial constraints. RAPT has been extended to incorporate a geometric modeling system and the a vision system to verify modeled part positions.

LM, NNS, SHARP, ACT (LIFIA)

Work by Latombe et al. at the "Laboratoire d'Informatique et d'Intelligence Artificielle (LIFIA)" at Grenoble, France, is on both mobile robots and assembly automation. Development of the LM system began in 1977 [LLLM84] with the objective of developing a manipulator level programming language. It grew into a robot programming environment incorporating teaching, simulation (including random effects), programming by specifying feature relations, and concurrently executing program blocks. Probabilistic uncertainty models have been used to model observed features [Cro87] and to represent and propagate geometric uncertainties through robot actions [MA89].
The SHARP system [LPT86] takes into account the interdependencies between grasping, transfer motions and part mating. Experience from SHARP and Handey (see below) was incorporated in the ACT robot programming environment [MPT91]. The automatic use of a vision system for verifying the outcome of manipulations is described in [Gor91].

[PTP88] propose a planning system using a set-oriented uncertainty model. The relations between objects are modeled by trees, but not arbitrary graphs. Acting and sensing operations are modeled by add- and delete-lists as well as by procedures for the backward propagation of uncertainty preconditions. They allow to prove the correctness of robot programs and to introduce observations into the program to satisfy preconditions.

SRL/RODABAS (University of Karlsruhe)

The SRL (Structured Robot Language) is based on the PASCAL language, extended by robotics related commands and data structures [Blu85, Rem87]. An SRL program is compiled, generating IRDATA code\(^3\) to be executed by a robot controller. An online database RODABAS based on the frame concept of AI keeps track of the existence and relations of objects.

Current work at Karlsruhe is on the two-armed mobile robot KAMRO [Hö92]. An online planner handles the allocation and collision avoidance of the two arms, and the regrasping of parts. Uncertainties are not modeled.

Handey (MIT)

Work by Brooks, Lozano-Pérez and others covers a broad range of topics, including model-based vision [Bro84], interpretation of sparse range data [GLP86, LPGW87], automatic robot programming [LPB86], robot motion planning and grasp planning.

The Handey robot programming system [LPJM+87] is able, using a depth map, to locate a part in an unstructured pile of objects. It finds a grasp point and plans moves to grasp the part and place it at a given destination.

SPAR (Purdue University)

At Purdue, the work of Kak, Lee et al. has lead to the implementation of SPAR [HK90b, HK90a]. SPAR solves geometric goals associated with object level goals. When given an object level goal such as on(A,B), it analyzes geometric constraints to determine whether regrasping is necessary and whether the goal can be achieved at all. It uses a set-based measure of uncertainty to plan sensing for uncertainty reduction, verification and error recovery. Other work at Purdue University includes probabilistic uncertainty models for planning [SL91] and the consistent interpretation and fusion of multiple observations [TL90].

---

\(^3\)IRDATA is a virtual robot level programming language defined by the German VDI. It provides a standard interface to robot controllers.
Chapter 3

UNCERTAIN GEOMETRIC RELATIONS

This chapter shows how uncertain geometric relations are represented and manipulated. First, basic methods for representing geometric uncertainty and for combining the uncertainty of several relations are given. Then it is shown how geometric uncertainties can be interpreted and compared. Finally, principles from network theory are used to maintain consistent networks of uncertain relations and to determine the uncertainty between arbitrary nodes of a network.

3.1 UNCERTAIN TRANSFORMS

The location and orientation of objects in 3-space, also called their pose, can be described by homogeneous transforms [Pau81, Wol87]. The homogeneous coordinates of a point within a coordinate system are represented by the vector \( \mathbf{v} = [x \ y \ z \ 1]^T \). The homogeneous transform matrix \( T_i \) transforms the representation \( \mathbf{v} \) of a point in coordinate frame \( i \) to its representation in coordinate frame \( j \) (Fig. 3.1):

\[
T_i \mathbf{v} = \mathbf{v}^j
\]

The homogeneous transform matrix \( T \) can be written as

\[
T = \begin{bmatrix}
  n_x & o_x & a_x & p_x \\
  n_y & o_y & a_y & p_y \\
  n_z & o_z & a_z & p_z \\
  0 & 0 & 0 & 1
\end{bmatrix}
\]

(3.1)

where \( n, o, a \) are the descriptions of the unit vectors of frame \( i \) in frame \( j \), and \( p \) is the displacement of frame \( i \) with respect to frame \( j \). Multiplication by the transform matrix accomplishes the rotation of \( \mathbf{v} \) and the addition of \( p \) in a single operation. A transform can be specified by three translations in the direction of the \( x, y \) and \( z \) axes of the reference coordinate frame, and three rotations about these axes. We define a description vector \( D_i \) containing the values of translation and rotation \( D_i = [p_x \ p_y \ p_z \ \phi_x \ \phi_y \ \phi_z]^T \). When referring to motions in specific directions, we shall mean them to include rotations. If the order of rotations is specified in roll-pitch-yaw coordinates, the complete transformation consists of a rotation of \( \phi_x \) about axis \( x \),
followed by a rotation of $\phi_y$ about the resulting axis $y'$, followed by a rotation of $\phi_z$ about the final axis $z''$ and a translation to $[p_x \ p_y \ p_z]$. 

For describing the uncertainty of transform information, we use an approach similar to that of [DW88, SC86, SL91]: Differential motion vectors are modeled as jointly gaussian random vectors, which allows to manipulate uncertainties regardless of their reference coordinate frames. There follows a summary of methods for transforming, propagating and integrating geometric uncertainty.

### 3.1.1 Representing uncertainty

We describe an uncertain transform (UT) by the sequence of a nominal transformation and a differential transformation. The nominal transformation is described by its transformation matrix $T$ or the corresponding description vector $D$. The differential transformation is described by a differential translation and rotation transformation $\Delta$ or the corresponding differential motion vector $d$, which have the form [Pau81]:

$$
\Delta = \begin{bmatrix}
0 & -\delta_z & \delta_y & d_x \\
\delta_z & 0 & -\delta_x & d_y \\
-\delta_y & \delta_x & 0 & d_z \\
0 & 0 & 0 & 0
\end{bmatrix}
$$

$$
d = \begin{bmatrix}
d_x \\
d_y \\
d_z \\
\delta_x \\
\delta_y \\
\delta_z
\end{bmatrix}
$$

(3.2)

Here, angular uncertainties are assumed to be small, and second-order and higher terms are neglected. Then $d$ is the description vector of the transformation matrix $\Delta + I$. We model the elements of the differential motion vector as jointly normal with zero mean and covariance matrix $\Lambda$. Its probability density is

$$
f_d(d) = \frac{1}{(2\pi)^{6/2}(\det\Lambda)^{1/2}}e^{-\frac{1}{2}d^T\Lambda^{-1}d}
$$

The uncertain transformation is completely described by $T$ and $\Lambda$. Note that $\Lambda$ is not the covariance of the description vector of $T$ [Mü92]. See [DW88] for the arguments for modeling errors as jointly normal random variables.
3.1. Uncertain Transforms

Notation: The UT relating frames $i$ and $j$ is described by the pair $(^i \mathbf{T}_i, ^i \Lambda_i)$. The differential transformation of a UT may be expressed in different frames (Fig. 3.2), and so may $^j \Lambda_j$. Therefore we use a leading subscript (e.g. $^j \Lambda_j$) to indicate that frame. If the leading subscript matches the leading superscript, it may be omitted. We shall call $^j \Lambda_p$ the uncertainty (matrix) of $^i \mathbf{T}_p$ (in frame $j$), and its inverse $^j \mathbf{Y}_p$ the information (matrix).

3.1.2 Transforming uncertainty

Let the uncertain pose of frame $p$ with respect to frame $i$ be given by the transform $(^i \Delta + I)^i \mathbf{T}_p$. What is the uncertain pose of frame $p$ with respect to another frame $j$? In other words, what are $^j \mathbf{T}_p$ and $^j \Lambda_p$?

The transform $^j \mathbf{T}_p$ is obtained immediately from the transform $^j \mathbf{T}_i$ which relates the frames $i$ and $j$:

$$^j \mathbf{T}_p = ^j \mathbf{T}_i ^i \mathbf{T}_p$$

The uncertainty $^j \Lambda_p$ is obtained by applying the following theorem for transforming random variables:

**Theorem 1** [DW88] If $^i \mathbf{v}$ is a random vector with mean $^i \mathbf{v}$ and variance $^i \Lambda_{\mathbf{v}}$, and if the function $^j F_i$ maps $^i \mathbf{v}$ to $^j \mathbf{v}$ as $^j \mathbf{v} = ^j F_i (^i \mathbf{v})$, then the mean and variance of $^j \mathbf{v}$ are

$$^j \mathbf{v} = ^j F_i (^i \mathbf{v})$$

$$^j \Lambda_{\mathbf{v}} = \left( \frac{\partial ^j F_i}{\partial ^i \mathbf{v}} \right) ^i \Lambda_{\mathbf{v}} \left( \frac{\partial ^j F_i}{\partial ^i \mathbf{v}} \right)^T$$

(3.3)

The term $(\partial ^j F_i / \partial ^i \mathbf{v})$ is called the Jacobian $^j \mathbf{J}_i$ of the function $^j F_i$. In our case, the Jacobian which transforms a differential change in frame $i$ into its representation in frame $j$ is found in the following way:

A differential change of pose may be expressed in the frame $i$ by $^i \Delta$ or in frame $j$ by $^j \Delta$. The two frames are related by $^j \mathbf{T}_i$ (Fig. 3.2). The complete transformations are $^j \mathbf{T}_i (^i \Delta + I)$ and $(^j \Delta + I)^j \mathbf{T}_i$. Equating them, simplifying and rearranging gives

$$^i \Delta = ^j \mathbf{T}_i^{-1} ^j \Delta ^j \mathbf{T}_i$$

(3.4)

Based on this, the differential motion vector $^i \mathbf{d}$ corresponding to $^i \Delta$ can be found directly from $^j \mathbf{d}$ as [Pau81, DW88]

$$^i \mathbf{d} = \begin{bmatrix} n^T & (p \times n)^T \\ o^T & (p \times o)^T \\ a^T & (p \times a)^T \end{bmatrix}$$

$$^j \mathbf{d} = ^{j} \mathbf{J}_i^{-1} ^j \mathbf{d}$$

(3.5)
where \( n, o, a \) and \( p \) are obtained from the transformation matrix as in (3.1) Since the covariance is given in frame \( i \), the Jacobian sought is

\[
\mathbf{J}_i = \begin{bmatrix}
    n & o & a \\
    0 & 0 & 0 \\
    (p \times n) & (p \times o) & (p \times a)
\end{bmatrix}
\]

Defining the \( 3 \times 3 \) rotation matrix \( r = [n \ o \ a] \), and the \( 3 \times 3 \) magnification matrix \( m = [(p \times n) \ (p \times o) \ (p \times a)] \), the Jacobian and its inverse can be written as

\[
\mathbf{J} = \begin{bmatrix}
r & m \\
0 & r
\end{bmatrix}, \quad \mathbf{J}^{-1} = \begin{bmatrix}
r^T & m^T \\
0 & r^T
\end{bmatrix}
\]

Summary: The uncertain pose of frame \( p \) with respect to frame \( i \) is given by the transform \((i\Delta + I)iT_p\). It is described by the pair \((iT_p, i\Lambda_p)\), where \( i\Lambda_p \) is the covariance matrix of the differential motion vector describing \( i\Delta \).

When changing frames, the uncertain pose of frame \( p \) with respect to frame \( j \) is described by

\[
\begin{align*}
\mathbf{j}T_p &= \mathbf{j}T_i iT_p \\
\mathbf{j}\Lambda_p &= \mathbf{j}J_i i\Lambda_p jJ_i^T
\end{align*}
\]

3.1.3 Propagating uncertainty: Compounding

Given two UTs \((iT_i, i\Lambda_i)\) and \((iT_p, i\Lambda_p)\), what is \((iT_p, j\Lambda_p)\) (Fig. 3.3)? Clearly, \( jT_p = jT_i iT_p \). The total uncertainty of frame \( p \) with respect to frame \( j \) may be inferred by transforming the known uncertainty of frame \( p \) with respect to frame \( i \) to frame \( j \), and adding the uncertainty of \( i \) with respect to \( j \) [DW88]:

\[
\mathbf{j}\Lambda_p = \mathbf{j}\Lambda_i + \mathbf{j}J_i i\Lambda_p jJ_i^T
\]

This relation holds only if the differential motion vectors characterized by \( i\Lambda_i \) and \( i\Lambda_p \) are statistically independent, i.e. if their covariance (or cross uncertainty) is zero. It implies that the cross uncertainty of the differential motion vectors associated with \( i\Lambda_i \) and \( j\Lambda_p \) is nonzero (similarly for \( i\Lambda_p \) and \( j\Lambda_p \)).

\footnote{[DW88] uses a different convention, the Jacobian used there is the inverse of the one defined here.}
3.1. Uncertain Transforms

If, after computation of the new UT from \( j \) to \( p \), the two old ones are discarded, then it is sufficient to keep \( j T_p \) and \( j T_p \). This operation of replacing a series of UTs by a single UT shall be called *compounding* [SC86].

3.1.4 Integrating uncertainty: Merging

Let the relations of a set of frames be described by a set of UTs. The frames and relations form a directed graph (or network). When sensors are applied, making observations, then additional UTs are created. An update of relations must be found which integrates the observations and leaves the network consistent: The product of transformations around any loop of the network must be the identity transform. A general solution to this problem can be found in [DW88] and is derived by different means in section 3.3.2. Here we show how to handle the two elementary tasks:

- Combining the information of UTs relating a pair of frames.
- Making the transforms around a loop of UTs consistent.

Combining information

Given two observations of the same relation, what is the uncertainty of the combined observations? Let the differential motion vectors expressing the difference between each of the observations and some common reference transformation be modeled as jointly normal, independent random vectors with mean \( d_1, d_2 \) and covariance matrices \( \Lambda_1, \Lambda_2 \). Then the combined maximum likelihood estimate has [DW88] mean vector

\[
\mathbf{d} = (\Lambda_1^{-1} + \Lambda_2^{-1})^{-1}(\Lambda_1^{-1}d_1 + \Lambda_2^{-1}d_2)
\]

and covariance matrix

\[
\Lambda = (\Lambda_1^{-1} + \Lambda_2^{-1})^{-1}
\]

(3.7)

(3.8)

If the transforms are consistent, then \( d_1 = d_2 = 0 \), and only (3.8) is of interest. The new UT determined by \( \mathbf{d} \) and \( \Lambda \) can replace the two old ones. This operation shall be called *merging* [SC86].
Making a loop of UTs consistent

Consider the observation of an object feature. It is represented by a new UT that forms a loop with the UTs of object, feature and sensor poses. What is the updated object pose and uncertainty, taking into account the a priori object pose and the uncertainties of the observation?

The problem may be restated for a three-node loop as: Given a sequence of uncertain transforms $T_1T_2T_3 \neq I$, with information matrices $\Upsilon_1$, $\Upsilon_2$, $\Upsilon_3$ described in the same arbitrary but fixed coordinate frame, find small changes in the transforms so that they form a loop

$$\hat{T}_1\hat{T}_2\hat{T}_3 = (\Delta_1 + I)T_1(\Delta_2 + I)T_2(\Delta_3 + I)T_3$$

minimizing the loss function

$$L = d_1^T\Upsilon_1 d_1 + d_2^T\Upsilon_2 d_2 + d_3^T\Upsilon_3 d_3,$$

where the $d_i$ are the differential motion vectors describing the differential translation and rotation transformations $\Delta_i$, which are the $\Delta_i$ transformed to the same common coordinate frame as the information matrices.

We transform the condition (3.10) to a condition on the $d_i$: Expanding and ignoring second-order terms of $\Delta_i$, we get

$$I = T_1T_2T_3 + \Delta_1T_1T_2T_3 + T_1\Delta_2T_2T_3 + T_1T_2\Delta_3T_3.$$

Postmultiplying by $(T_1T_2T_3)^{-1}$ gives

$$(T_1T_2T_3)^{-1} = I + \Delta_1 + T_1\Delta_2(T_1)^{-1} + T_1T_2\Delta_3(T_1T_2)^{-1}$$

Applying (3.4) gives the differential translation and rotation transformations in a common frame (which happens to be frame 1). Moving over the identity matrix, we get

$$(T_1T_2T_3)^{-1} - I = \Delta = \Delta_1 + \Delta_2 + \Delta_3$$

which means that we may also equate the associated differential motion vectors (see (3.2))

$$d_{\text{total}} = d_1 + d_2 + d_3$$

We minimize the loss function (3.11) subject to this constraint on the $d_i$ by applying lagrangian multipliers, obtaining

$$\Upsilon_1 d_1 = \Upsilon_2 d_2 = \Upsilon_3 d_3$$

This is intuitively satisfying. Solving (3.12) and (3.13) for $d_i$ gives

$$d_i = \Lambda_i (\Lambda_1 + \Lambda_2 + \Lambda_3)^{-1}d_{\text{total}} \quad i = 1 \ldots 3$$
3.1. Uncertain Transforms

Figure 3.4: *Recursive updating for making loop transforms consistent*

which can easily be generalized to a loop of $n$ transformations.

The update of loop transforms can also be implemented recursively: Consider a loop, with nodes labeled as in Fig. 3.4. Let $\Lambda_n = \sum_{i=2}^{n} i^{-1} \Lambda_i$ be the uncertainty of the relation resulting from compounding the relations via nodes 2, 3, … $n-1$. The update in the location of node $n$ relative to node 1 is determined by the loop consisting of this compounded relation and the relation from node $n$ to 1, which has uncertainty $\Lambda_n$.

In the next step, the change of location of node $n-1$ is determined by distributing the change over the relations from node 1 to $n-1$ and $n-1$ to $n$, weighted by $\Lambda_{n-1} = \sum_{i=2}^{n-1} i^{-1} \Lambda_i$ and $\Lambda_n$. This step is repeated for nodes $n-2, n-3, \ldots, 2$.

3.1.5 Interpretation of covariance matrices

Given an arbitrary covariance matrix of a UT, its interpretation is not obvious. It is desirable to have a point of view from which it has a simple, intuitively understood interpretation. Consider a diagonal covariance matrix: The errors in the six directions are independent, and each diagonal element is a measure of pose uncertainty in a specific linear or angular direction.

A covariance matrix resulting from repeated transformations and integration operations will, in general, not be diagonal. If only position (and no rotation) errors are considered, their covariance is a $3 \times 3$ matrix. The axis of the uncertainty ellipsoid can be found by eigenvalue analysis and visualized in $\mathbb{R}^3$ [MB88, VMT90]. But our covariance matrices are of size $6 \times 6$, and $\mathbb{R}^6$ is difficult to visualize. Given a covariance matrix, it is desirable to find a transformation to a frame within $\mathbb{R}^3$ in which the covariance may be readily interpreted by a human or by a reasoning system based on human reasoning strategies.

Covariance matrices are real, symmetric and positive definite, and remain so when merging and compounding UTs. Since real symmetric matrices are orthogonally diagonalizable, we can write for any uncertainty $\Lambda$

$$\Lambda = S \Lambda S^T$$

with $SS^T = I$

where $\Lambda$ is diagonal and $S$ is a matrix of eigenvectors of $\Lambda$. Repeated eigenvalues of $\Lambda$ must have different eigenvectors. This general diagonalization is of no immediate
use, since there is no ready interpretation of the transform matrix $S$. We therefore examine the transformation of uncertainties in detail: Given an uncertainty matrix

$$\Lambda = \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{bmatrix}$$

we have seen that its representation in another frame is

$$\vartheta\Lambda = J \Lambda J^T$$

(3.15)

where $J$ has the structure

$$J = \begin{bmatrix} r & m \\ 0 & r \end{bmatrix}$$

The blocks of $\vartheta\Lambda$ are found to be

$$\begin{align*}
\vartheta\Lambda_{11} &= r \Lambda_{11} r^T + m \Lambda_{21} r^T + r \Lambda_{12} m^T + m \Lambda_{22} m^T \\
\vartheta\Lambda_{12} &= r \Lambda_{12} r^T + m \Lambda_{22} r^T \\
\vartheta\Lambda_{21} &= r \Lambda_{21} r^T + r \Lambda_{22} m^T \\
\vartheta\Lambda_{22} &= r \Lambda_{22} r^T
\end{align*}$$

(3.16) \hspace{1cm} (3.17) \hspace{1cm} (3.18) \hspace{1cm} (3.19)

The last equation shows that the angular uncertainty is independent of the translation part of the transformation. The associated uncertainty ellipsoid has no preferred location in space. Its orientation is determined by a rotation that makes $\vartheta\Lambda_{22}$ diagonal. As $\Lambda$ is symmetric and positive definite, so are $\Lambda_{11}$ and $\Lambda_{22}$, since they are principal minors of $\Lambda$. From (3.19) we can make $\vartheta\Lambda_{22}$ diagonal by choosing $r = v^{-1}$, where $v$ is the matrix of orthonormal eigenvectors of $\Lambda_{22}$. Then (3.19) becomes

$$\vartheta\Lambda_{22} = v^T \Lambda_{22} v$$

and $\vartheta\Lambda_{22}$ is the diagonal matrix containing the eigenvalues of $\Lambda_{22}$.

The orientation of the ellipsoid of angular uncertainties can now be displayed and interpreted in $\mathbb{R}^3$. Its main axes indicate the axes of rotation about which uncertainty is maximal and minimal. Their directions are of interest when planning sensing operations.

The linear uncertainty $\vartheta\Lambda_{11}$ depends on $m$ and therefore on translation. However, the linear part of the corresponding information matrix does not. This is seen e.g. by substituting (3.17) into (3.16) and applying formulas for blockwise matrix inversion. It is also seen by substituting the information matrices $\vartheta\Upsilon = \vartheta\Lambda^{-1}$ and $\Upsilon = \Lambda^{-1}$ into (3.15), which gives the equation for transforming information:

$$\vartheta\Upsilon = J^{-T} \Upsilon J^{-1}$$

Partitioning the matrices in the same way as the uncertainties gives a set of equations dual to Eqs. 3.16 through 3.19, and in particular

$$\vartheta\Upsilon_{11} = r \Upsilon_{11} r^T$$
3.2. Comparison of uncertainties

This shows that the ellipsoids of linear information have no preferred location in space. They can be visualized in the same way as the ellipsoids of angular uncertainty.

These results have an analogy in the analysis of elastically supported rigid bodies [LP92]. Uncertainty and information matrices correspond to compliance and stiffness matrices. Mechanical and energy conservation arguments are used to determine linear compliance and angular stiffness ellipsoids. Forces (or torques) in the direction of the ellipsoid axes result in displacements parallel to (or rotations about) the same axes.

3.2 COMPARISON OF UNCERTAINTIES

When the feasibility of an operation is assessed, the actual uncertainty of some relation must be smaller than a given maximum uncertainty. This is the same as saying that the actual information must be larger than a given minimum information. If the actual information is not large enough, it is desirable to know what kind of additional information is required.

3.2.1 Verifying the sufficiency of given information

What exactly is meant by "larger uncertainty"? A covariance matrix can be interpreted in terms of a confidence ellipsoid around the mean of its random vector. The boundary of the ellipsoid is determined by values with constant probability density, and it enfolds the set of values that, with some given confidence, contains the random vector.

We shall say that, for covariance matrices $A$ and $B$, the uncertainty described by $A$ is greater than that of $B$ if the confidence ellipsoid determined by $B$ is completely contained in the confidence ellipsoid determined by $A$. This is the same as stating, for covariance matrices $A, B \in M_n$,

$$e^{-x^TA^{-1}x} > e^{-x^TB^{-1}x} \quad x \in \mathbb{R}^n$$

(3.20)

The confidence regions are bounded by the ellipsoids

$$x^TA^{-1}x = x^TB^{-1}x = \text{const.}$$

For the following argument, we must first introduce the positive semidefinite ordering of Hermitian matrices. The ordering is defined by [HJ88, p. 469]:

**Definition 1** Let $A, B \in M_n$ be Hermitian Matrices. We write $A \succeq B$ if the Matrix $A - B$ is positive semidefinite. Similarly, $A \succ B$ means that $A - B$ is positive definite.

This relation is a partial order, i.e. there exist Hermitian matrices such that neither $A \succeq B$ nor $B \succeq A$. For positive definite matrices $A$ and $B$, the properties that we shall use are:

---

$^2$The following discussion assumes that the uncertainties being compared are expressed in the same coordinate frame.
1. $A \succeq B$ if and only if $B^{-1} \succeq A^{-1}$

2. If $A \succ B$ and $\det(x) \neq 0$ then $x^T A x > x^T B x$ for $x \in \mathbb{R}^n$

Using these properties, it is easy to prove that $A \succ B$ if and only if (3.20) holds. The positive semidefinite ordering therefore is exactly the desired ordering of uncertainties. For example, the first property can be interpreted as "if the uncertainty $A$ is larger than $B$, then the information $A^{-1}$ is smaller than $B^{-1}$".

For deciding whether the uncertainty $A$ is greater than $B$, one now may check whether the difference $A - B$ is positive definite. This can be done e.g. by verifying whether all its eigenvalues are positive.

3.2.2 Determining what information should be acquired

If it cannot be shown that $A \succ B$, it is useful to know how the uncertainty $B$ should be reduced in order to make the inequality hold. If $B \succ A$, then $B$ must be reduced in all directions. If not, then there is sufficient information in at least one direction, and insufficient information in orthogonal directions. The directions in which information is not sufficient are found by considering the eigenvalues of $A - B$: Each negative eigenvalue corresponds to a direction in which the ellipsoid $x^T A^{-1}x = 1$ lies inside $x^T B^{-1}x = 1$. The direction is given by the corresponding eigenvector.

Unfortunately, if we wish to use this information to guide the planning of sensing operations, we see that there exists no ready interpretation of these vectors: They lie in the space spanned by the six unit vectors of differential motion. If a vector lies only in the subspace corresponding to translations (or rotations), then it is obvious that the position of the corresponding object in that direction (or its rotation about that axis) must be observed. But how to interpret a vector that does not lie in one subspace alone? We get around this problem by constructing an information matrix $Y_{Obs}$ which is larger than the needed information, and which has decoupled translation and rotation submatrices $Y_{ObsT}$ and $Y_{ObsR}$. Its eigenvectors therefore are restricted to the translation and rotation subspaces. After stating the problem, we show the construction of the solution for the case of three-dimensional uncertainties and then prove its validity for the $n$-dimensional case.

Since an observation leads to the addition of information matrices, we state the problem in terms of information instead of uncertainties: Given some minimum required information $Y_{Min}$ and the currently available information $Y_{Cur}$, where $Y_{Cur} \not\succeq Y_{Min}$, we wish to determine some observation information $Y_{Ob}$ such that

$$Y_{Ob} = \begin{bmatrix} Y_{ObsT} & 0 \\ 0 & Y_{ObsR} \end{bmatrix} \text{ and } Y_{Ob} + Y_{Cur} \succeq Y_{Min}$$

Introducing $Y_N = Y_{Min} - Y_{Cur}$, the additional information needed, the last condition can be stated as

$$Y_{Ob} \succeq Y_N$$

(3.21)
3.2. Comparison of uncertainties

Figure 3.5: Minimum needed observation information. The ellipsoid $x^T Y_{\text{obs}}^{-1} x = 1$ circumscribes the cylinder with generating surface $x^T Y_{N_Y}^{-1} x = 1$, which in turn contains the required minimally needed observation information ellipsoid.

Based on a geometric argument for the three-dimensional case, we may construct an $\Upsilon_{\text{obs}}$ which satisfies this condition. Let the three dimensions correspond to the location $(x,y)$ and orientation $(\phi)$ of a 2-D object in the xy-plane. The needed information matrix is partitioned as

$$\Upsilon_N = \begin{bmatrix} \Upsilon_{NT} & \Upsilon_{NC} \\ \Upsilon_{NT}^T & \Upsilon_{NR} \end{bmatrix}$$  \hspace{1cm} (3.22)

Consider the ellipsoid on which $x^T Y_N^{-1} x = 1$. Regardless of $\Upsilon_{NC}$, it can lies within the cylinder with the elliptical generating surface $x^T Y_{N_Y}^{-1} x = 1$ and with length $\sqrt{\Upsilon_{NR}}$ (which is a scalar) along the positive and negative $\phi$-axis (Fig. 3.5). The ellipsoid corresponding to $\Upsilon_{NC} = 0$ touches the cylinder in the middle of its plane surfaces and along its intersection with the xy-plane. Multiplying the ellipsoid’s main axis lengths by $\sqrt{2}$ gives an ellipsoid which completely encloses the cylinder. This corresponds to multiplying the information matrix describing the ellipsoid by 2. This is the required observation information $\Upsilon_{\text{obs}}$.

$$\Upsilon_{\text{obs}} = 2 \begin{bmatrix} \Upsilon_{NT} & 0 \\ 0 & \Upsilon_{NR} \end{bmatrix}$$  \hspace{1cm} (3.23)

We now show that this observation information satisfies (3.21) for arbitrarily sized information matrices: For $\Upsilon_N, \Upsilon_{\text{obs}} \in \mathcal{M}_n$ and $\Upsilon_{NT} \in \mathcal{M}_m$, with $n > m$, (3.21) holds, if

$$\Upsilon_{\text{obs}} - \Upsilon_N = \Upsilon_D$$

is positive semidefinite. From (3.22) and with $\Upsilon_{\text{obs}}$ chosen as in (3.23),

$$\Upsilon_D = \begin{bmatrix} \Upsilon_{NT} & -\Upsilon_{NC} \\ -\Upsilon_{NT}^T & \Upsilon_{NR} \end{bmatrix}$$
The eigenvalues of $\mathbf{Y}_D$ are the same as those of $\mathbf{Y}_N$, since $\mathbf{Y}_D$ can also be written as

$$\mathbf{Y}_D = \mathbf{P} \mathbf{Y}_N \mathbf{P}^T$$

with

$$\mathbf{P} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Therefore, if $\mathbf{Y}_N$ is positive semidefinite, so is $\mathbf{Y}_D$.

If $\mathbf{Y}_M \neq \mathbf{Y}_C$, i.e. if the currently available information is already sufficiently large in some directions, then $\mathbf{Y}_N$ is not positive semidefinite. In order to allow the above construction of $\mathbf{Y}_{\text{Obs}}$, $\mathbf{Y}_N$ it can be modified as follows: $\mathbf{Y}_N$ is transformed to diagonal form and the negative eigenvalues (corresponding to the directions with sufficient information) are set to zero. Transforming the resulting matrix back again gives $\mathbf{Y}_N'$ which is positive semidefinite.

Our choice of $\mathbf{Y}_{\text{Obs}}$ is only one of a continuum of choices that make $\mathbf{Y}_D$ positive semidefinite. In general,

$$\mathbf{Y}_D = \mathbf{Y}_{\text{Obs}} - \mathbf{Y}_N$$

$$= \begin{bmatrix} \mathbf{Y}_{\text{Obs}T} & 0 \\ 0 & \mathbf{Y}_{\text{Obs}R} \end{bmatrix} - \begin{bmatrix} \mathbf{Y}_{NT} & \mathbf{Y}_{NC} \\ \mathbf{Y}_{NC}^T & \mathbf{Y}_{NR} \end{bmatrix}$$

is positive definite, if and only if [HJ88, p. 472]

$$\mathbf{Y}_{\text{Obs}T} - \mathbf{Y}_{NT} \succeq 0$$

and

$$\mathbf{Y}_{\text{Obs}R} - \mathbf{Y}_{NR} \succ \mathbf{Y}_{NC}^T (\mathbf{Y}_{\text{Obs}T} - \mathbf{Y}_{NT})^{-1} \mathbf{Y}_{NC}$$

These conditions show that, within certain bounds, an increase in $\mathbf{Y}_{\text{Obs}T}$ enforces an increase in $\mathbf{Y}_{\text{Obs}R}$. There is a tradeoff between the required observation information along linear and angular axes. Our choice of $\mathbf{Y}_{\text{Obs}}$ minimizes the volume of the observation uncertainty ellipsoid satisfying these conditions.

### 3.3 NETWORKS OF UNCERTAIN RELATIONS

We have seen how to propagate uncertainty and integrate UTs consistently in pairs or loops of UTs. We shall proceed to show what problems arise in an assembly environment and how they are solved by extending the above methods to handle arbitrary networks of UTs.

#### 3.3.1 Requirements of assembly systems modeling

Consider the situation depicted in Fig. 3.6. The UTs describing object poses may form a directed graph as shown. The robot system can now perform the following actions, whose effect on the system's knowledge of the world must be reflected in its workspace model: A sensor may make observations of part features, giving information about the relation between parts or between a part and a sensor. An actuator may grasp parts, move them (together with any affixed parts) and join them to one another. A planner may compute the relative uncertainty between object poses and assess the feasibility of a sensing or acting operation. In terms of UT manipulation, these operations require the ability to
Figure 3.6: An assembly environment and its graph of relations
Chapter 3. Uncertain geometric relations

Figure 3.7: A bridge and an equivalent network. After applying a delta-star transform to the lower triangle, the uncertainty between nodes A and B can be determined by merging and compounding.

- create an UT between arbitrary nodes of the graph and to update the pose data of the graph consistently.
- delete an UT which is known to have become invalid due to some motion of objects. Information that remains valid may be propagated into other UTs.
- find the uncertainty in the relative pose of arbitrary nodes of the graph.

Consistent updating and uncertainty computation can be dealt with by considering the graph of UTs alone. They are addressed in the following sections. The problem of deleting UTs involves knowledge about the manipulation of objects. It is discussed in section 4.1.4.

3.3.2 The network approach

The procedures for compounding and merging UTs suggest the analogy with electric circuit theory, which was also noted by [SC86]. There the analogy was used to solve the following problem: In Fig. 3.6, the uncertainty between frame C and the world frame cannot be found by merging and compounding operations alone, since it contains a bridge network. The analogy suggests a delta-star transformation, creating an equivalent network which then can be reduced by merging and compounding (Fig. 3.7). Unfortunately, arbitrary networks can still not be reduced with this approach. But if the analogy is carried further, concepts from network theory can be applied to the general problem.

The mechanical network analogy

The behavior of a network of UTs is also analogous to that of a mechanical system. Consider a network of rigid bodies connected by compliant couplings. A coupling can be modeled as a six degree of freedom spring and is described by a stiffness matrix. The following analogies exist between UTs and compliant couplings:
• A stiffness matrix (and its inverse, a compliance matrix) is real, symmetric and positive definite.

• The transformation of a stiffness matrix into another frame of representation follows the same laws as the transformation of an information matrix, using the Jacobian of the transform [CB91].

• The stiffness of a parallel or a serial connection of two compliant couplings is determined in the same way as the information of a parallel or a serial connection of two UTs. This is obvious for single degree of freedom springs. For the general case see [GD90].

• The conservation laws analogous to the Kirchhoff voltage and current laws hold in both types of networks.

The last point remains to be demonstrated. Consider a network of UTs with \( n \) nodes and \( m \) arcs \((n \leq m)\). Its topology is described in the usual manner: Define \( M \) to be the directed network incidence matrix of size \( 6n \times 6m \) whose submatrices \( M_{ij} \) take the values

\[
M_{ij} = \begin{cases}
I & \text{if arc } j \text{ leaves node } i \\
-I & \text{if arc } j \text{ enters node } i \\
0 & \text{otherwise}
\end{cases}
\]

with \( I \) being the \( 6 \times 6 \) identity matrix. Define \( C \) to be a \( 6r \times 6m \) basic loop matrix of the network, where \( r = m - n + 1 \) is the number of fundamental loops of the network.

The submatrices \( C_{ij} \) take the values

\[
C_{ij} = \begin{cases}
I & \text{if arc } j \text{ is in loop } i, \text{ and their directions agree} \\
-I & \text{if arc } j \text{ is in loop } i, \text{ and their directions oppose} \\
0 & \text{otherwise}
\end{cases}
\]

Absolute and relative node displacements are assumed to be of small magnitude and relative to some reference state of the network, such that their relations can be linearized. Let the absolute pose of node \( i \) be described by the differential motion vector \( x_i \) with respect to the node's fixed reference pose. Similarly, let the relative pose corresponding to arc \( i \) be described by \( d_i \). Let all \( x_i, d_i \) be described in the same coordinate frame. With \( x = [x_1^T, \ldots, x_n^T]^T \) and \( d = [d_1^T, \ldots, d_m^T]^T \), it follows from (3.12) (since \( d_{\text{total}} = 0 \)), that in a consistent network arc and node displacements are related by

\[
d = M^T x
\]

and that

\[
Cd = 0
\]

The last equation corresponds to Kirchhoff's voltage law, stating that the sum of displacements around any loop is zero.

For the dual law, we use a result from [DW88] that shows how to integrate observations into an existing network of UTs. We state it here in a less complicated form, and with the important distinction that we represent small transforms by differential motion vectors, and not by differences in description vectors [Mü92].
Figure 3.8: Integrating observation networks. The a priori network (left) and the network formed by observations (right) are described by $T$ and $T_2$. Matching equally labeled nodes requires changes in node locations, weighted by the information on arcs.

Consider a sequence $z_1, \ldots, z_k$ of observations of arcs of a network. If several observations of the same arc are made, they can be merged such that just one new arc remains to be integrated into the network. Given the a priori node locations $x$ and the observed arc poses $z$, the problem is to find new estimates $\hat{x}$ of node locations that give a consistent integration of the observations with the prior network (Fig. 3.8).

Let $M$ and $C$ describe the network containing the arcs known prior to observation as well as those introduced by the observations. Let $T$ be the block diagonal matrix of a priori arc information, where the information corresponding to arcs introduced by the observations is zero. Let $z$ be the vector of observations, with entries corresponding to arcs which were not observed set to zero. Let $T_2$ be the matrix of observation arc information, where the information corresponding to arcs that are known a priori but are not observed is zero.\(^3\) Let the initial state of the network be the reference state with $x = 0$ and $d = 0$.

The changes $\hat{d}$ in arc transforms are chosen to minimize the quadratic loss function

$$L(\hat{d}) = \frac{1}{2} \hat{d}^T T \hat{d} + \frac{1}{2} (\hat{d} - z)^T T_2 (\hat{d} - z)$$

subject to the constraint

$$C\hat{d} = 0$$

An intermediate result in [DW88] shows that the solution satisfies

$$MT\hat{d} + MT_2(\hat{d} - z) = 0$$

(3.27)

Interpreting this equation in terms of mechanical structures, we see that $T\hat{d}$ corresponds to the force applied to a compliant coupling with stiffness $T$ which results in the relative displacement $\hat{d}$ of its endpoints. Row $i$ of the term $MT\hat{d}$ then corresponds to the sum of forces applied at node $i$. Likewise, $MT_2(\hat{d} - z)$ gives forces acting on arcs.

\(^3\)This way of defining $M$, $T$, $x$, $T_2$, $z$ avoids cluttering the equations with auxiliary matrices that map a priori and observed arcs onto one another.
3.3. Networks of uncertain relations

the network corresponding to the observations. The entire equation states that, for a network in a state of equilibrium, the sum of forces at each node is zero.

Therefore, for a network of UTs, we postulate a quantity that corresponds to forces and that obeys the same conservation law, corresponding to Kirchhoff’s current law.

We shall neither name nor use this quantity explicitly, since we are only interested in justifying the analogy between networks of UTs and compliant couplings.

In mechanics, the equivalent of (3.27) is arrived at by e.g. the potential energy approach, in which a state of equilibrium is found that minimizes the potential energy of a system. The expression for the potential energy in a compliant structure has the same form as the loss function in (3.26).

Substituting (3.24) in the term \( \Phi \), it becomes \( \Phi = M^T \Phi \). The term

\[
\Phi = M^T
\]

is also found in finite element theory, where it is called the global or structural stiffness matrix, and where the equation

\[
M^T \Phi \hat{x} = f_{ext}
\]

relates node displacements to external forces. In electric circuit theory, \( \Phi = M^T \) corresponds to the indefinite admittance matrix.

Definition 2 The Indefinite Information Matrix (IIIM) of a network of UTs is \( \Phi = M^T M \), where \( \Phi \) is the block diagonal matrix of UT information matrices and \( M \) is the network incidence matrix.

The IIIM can be determined directly from the graph of UTs by the following rules. The \( 6 \times 6 \) submatrices at position \((i, i)\) on the diagonal are the sum of the information matrices of all arcs entering or leaving node \( i \), and the submatrices at position \((i, j)\) are the negative sum of all arcs joining nodes \( i \) and \( j \). It immediately follows that the IIIM is symmetric and singular with rank \( 6(n - 1) \).

3.3.3 Consistent integration of observations

Returning to networks of UTs, we complete the argument of [DW88] by substituting (3.24) in (3.27) and rearranging, obtaining

\[
M(\Gamma + \Gamma_z)M^T \hat{x} = M \Gamma_z z
\]

which allows to determine the new estimates of node locations \( \hat{x} \) from the observed arc transformations \( z \). The term \( M(\Gamma + \Gamma_z)M^T \) is the IIIM of the updated network.

4In the remainder of this section, indices \( i \) and \( j \) denote the position of vectors and matrices within larger, regularly partitioned matrices
Figure 3.9: Uncertainty between two nodes. For a given IIM, deleting rows and columns associated with node $j$ amounts to "grounding" the node. In a mechanical structure, the displacement of any node $i$ when applying a force $f_i$ to it is determined by the stiffness of node $i$ relative to node $j$.

3.3.4 Determining relative uncertainty

So far, the analogies between networks have been intellectually pleasing but have not offered any new results. We now apply them to the so far unsolved problem of determining the relative uncertainty between arbitrary nodes of a network.

Since the information of a UT corresponds to the stiffness of a mechanical element, the uncertainty between any two nodes in a network of UTs corresponds to the compliance encountered when applying a force between these two nodes in the corresponding mechanical network (Fig. 3.9).

Thinking in terms of mechanical networks, (3.28) which relates external forces to node displacements can be written as

$$\Phi \mathbf{x} = \mathbf{f}_{\text{ext}}$$

(3.30)

The stiffness encountered when applying a force between two arbitrary nodes is just the relation between that force and the resulting displacement of the two nodes. Let the two nodes be labeled $i$ and $j$ and let the applied force be

$$\mathbf{f}_{\text{ext}} = [0, \ldots, 0, f_i, 0, \ldots, 0, f_j, 0 \ldots, 0]^T$$

with $f_i + f_j = 0$. We now need to know $\delta \mathbf{x} = \mathbf{x}_i - \mathbf{x}_j$.

The set of equations represented by (3.30) is linearly dependent, since $\Phi$ is of rank $6(n - 1)$. We are free to choose one node location arbitrarily, e.g. $x_j = 0$, and to ignore six equations, e.g. the rows corresponding to $f_j$. Deleting the corresponding rows and columns $6j \ldots 6j + 5$ of $\Phi$ gives the matrix $[\Phi_j^T]$, which we shall call definite
3.4 Realization of network functions

The reduced set of equations is

\[
\begin{bmatrix}
X_1 \\
\vdots \\
X_i \\
\vdots \\
X_{j-1} \\
X_j+1 \\
\vdots \\
X_n
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
\vdots \\
0 \\
f_i \\
\vdots \\
0
\end{bmatrix}
\tag{3.31}
\]

The compliance/uncertainty of interest is \( \Lambda_\alpha \) such that \( x_i = \Lambda_\alpha f_i \). It immediately is seen that \( \Lambda_\alpha \) is the 6 \times 6 submatrix at position \((i, i)\) of \( [\Phi^T_j]^{-1} \).

3.4 REALIZATION OF NETWORK FUNCTIONS

We have shown how to integrate observations and compute relative uncertainties over simple sequences and arbitrary networks of relations. This section presents the combination of these procedures as implemented in the ARGUS system.

Computing uncertainties by merging and compounding relations is fast, but does not allow to treat arbitrary networks. The network approach is general, but solving the network equations by direct matrix inversion is inefficient. Sparse matrix techniques for representation and computation may be more efficient, but the sparsity pattern of an IIM does not suggest an simple solution, as e.g. a banded matrix would. The following approaches were examined:

1. Keeping the inverse of the DIM in memory and updating it as the DIM changes, using formulas for incremental and blockwise matrix inversion. In this way, the inversion of the complete DIM is avoided. The efficiency increases if uncertainties relative to the same node need to be computed frequently. It decreases if the network topology, uncertainties or the reference node change frequently.

2. Reducing a DIM by node suppression, exploiting the sparsity of the right hand side of (3.31). This is only applicable for finding relative uncertainties, but not for integrating relations.

3. A mixed approach, combining network simplification by compounding and merging with the solution of network equations. In order to determine the relative uncertainty of two nodes, a network of relations is reduced to a single relation. The integration of a new relation is done by recursively updating the representation of such a reduced network.

The last approach was implemented since it is applicable to both problems and because the network simplification part is efficient to implement in Prolog. It takes sparsity into account implicitly: The sparsity pattern is the result of network topology, and
simplifying the network by merging and compounding gives a network whose IIM is smaller and less sparse, and can be inverted with less computational effort. After some graph theoretic preliminaries and an overview of the network reduction algorithm, its components will be described in detail.

Graph theory preliminaries

We use the following concepts of graph theory [ST81]: A graph consists of a finite set $V$ of vertices and a finite set $E$ of edges. Each edge is identified with a pair of vertices. A walk in a graph is a finite alternating sequence of vertices and edges, beginning and ending with vertices. A walk is open if its end vertices are distinct, otherwise it is closed. A walk is a trail if all its edges are distinct. An open trail is a path if all its vertices are distinct. A closed trail is a circuit if all its vertices except the end vertices are distinct. Two vertices are connected if there exists a path between them. A graph is connected if every pair of its vertices is connected. A graph can be partitioned into components: Each component of a graph is a connected subgraph in which no vertex is connected to a vertex of another component. A cut-vertex is a vertex whose removal increases the number of components of the graph. A biconnected graph is a graph with no cut-vertices. A biconnected component (or block) is a maximal biconnected subgraph of a graph.

In our context, we shall use the terms node and relation instead of vertex and arc. Based on above, we further define (Fig. 3.10)

- The path set $P_{ab}$ in a graph, defined by nodes $a$ and $b$, is the graph containing all paths connecting $a$ and $b$. 

---

**Figure 3.10:** A graph, pathsets and components. (1) The original graph. (2) The path set $P_{ab}$. (3) The components of $P_{ab}$. (4) The path set $P_{ac}$. It has exactly one component, i.e. itself.
3.4. Realization of network functions

- A path set component is a component of the graph obtained by removing the end nodes \( a \) and \( b \) from the path set \( \mathcal{P}_b \).
- A graph is called reducible, if relations or nodes can be removed by merging or compounding.

3.4.1 Network reduction

The outline of the complete procedure is as follows: Given a net of relations, and a pair of nodes \( \{a, b\} \), a representation for determining the relative uncertainty \( \Lambda_b \) and for integrating a new relation between \( a \) and \( b \) is established by the following algorithm (See Fig. 3.11):

1. Determine the path set \( \mathcal{P}_b \) and its components.
2. Compute the uncertainty of each relation \( R \in \mathcal{P}_b \) and its representation in a common coordinate frame.
3. Simplify each component of \( \mathcal{P}_b \) as far as possible by merging and compounding relations.
4. If the path set has been reduced to a single relation:
   → Stop.
   Else:
   → Determine the subnets which must be handled by the network approach. Use it to compute the uncertainty over the subnets, and replace the subnets by single arcs having that uncertainty. Go to Step 3.

This procedure will reduce any network to a single relation \( \mathcal{R}_b^{red} \) joining nodes \( a \) and \( b \). The uncertainty of the relation is the required uncertainty.

Finding all paths joining two given vertices

Determining \( \mathcal{P}_b \) is done using a modified depth-first search of the graph, starting at vertex \( a \) and keeping track of the path back to \( a \). Paths not leading to the goal node \( b \) are discarded. If the search backtracks over the start node, a new path set is generated. If, later on, the search encounters a node of a previously established path set, then the two path sets are merged.

Updating uncertainty representations

The subsequent uncertainty computations require that the uncertainties of all relations are represented in a common reference frame, usually the world frame. This representation depends on the transform to the reference. As transforms change when parts are manipulated, the uncertainties become invalid. We choose to update them after having found the path set for the current problem. This allows to limit the update effort to the path set.
Figure 3.11: Network simplification. (1) The original graph. (2) After finding the path set between nodes a and b. (3) After simplifying by merging and compounding. (4) After reducing biconnected components. Further merging and compounding reduces the net to a single relation.

The update is done by starting at the reference node, and at each node examining the not yet updated relations to the adjacent nodes. The relation's uncertainty in the reference frame is updated if either the reference frame has changed, or if the transformation along the path to the reference frame has changed, or if the relation itself has changed.

Simplifying a path set

The following two procedures for compounding and merging relations are applied to $^aP_b$ in sequence until neither of them is applicable anymore.

- Compounding: For any node $x$ in $^aP_b$ but not equal to $a$ or $b$, find the set of adjacent nodes. If this set contains exactly two nodes $\{y, z\}$, compute $^y\Lambda_x = ^y\Lambda_x + ^z\Lambda_z$ and $^yT_z = ^yT_x T_z$, delete $^yR_x$, $^zR_z$ and create a new relation $^yR_z$.

- Merging: For any pair of adjacent nodes $\{x, y\}$, find the set of relations joining them. If this set $\{^zR_y, \ldots, ^zR_y^n\}$ contains more than one relation, compute $^zA_y = \left(\sum_{i=1}^{n} (^zA_y)^i\right)^{-1}$, delete the found relations and create a single new relation $^zR_y$.

The resulting path set is either a single relation or contains an irreducible subgraph.

Reducing biconnected components

It is not necessary to apply the network approach to the entire simplified network. Consider Fig. 3.11(3). There it is sufficient to replace the irreducible bridge network
between nodes \(a\) and \(c\) by a single relation. The resulting network, as in Fig. 3.11(4),
can then be further simplified by merging and compounding. The problem of finding
the irreducible subgraphs can be solved using an algorithm for finding the biconnected
components of a graph:

- For each of the path set components of \(P_b\), determine the biconnected com­
  ponents and the cut-nodes.
- Each biconnected component has exactly two cut-nodes in common with the
  remainder of the path set, and is either a single arc or an irreducible subgraph of
  the path set component.
- In the latter case, compute the transformation and the uncertainty over the path
  set component, using the network approach. Delete the relations forming the
  irreducible subgraph and create a new relation joining the cut-nodes associated
  with the deleted subgraph.

The interesting part is the algorithm for finding the biconnected components of a
graph. [ST81] gives an algorithm, based on a depth-first search of the graph. It
is of complexity \(O(n + m)\), where \(n\) is the number of vertices and \(m\) is the number
of edges in the graph. For this work, the algorithm was extended to record the pair of
cut-nodes associated with each biconnected component.

Recursive network representation

Recursive algorithms are used for both network simplification (when computing un­
certainties) and expansion (when integrating observations). This motivates a recursive
data structure for representing the network and the reduction steps. It is defined by
the following syntax:

\[
\begin{align*}
\text{Relation} & := ("\text{' Node1, Node2 '}", \text{Expansion}, \text{Traf}, \text{Unc}) \\
\text{Expansion} & := \\
& \text{'ser(' Relation ', Relation')'} \\
& \text{'par([Relation ')', Relation '])'} \\
& \text{'net([Relation ')', Relation '])', Node1'} \\
& \text{'inv' Relation} \\
& \text{'simple'}
\end{align*}
\]

3.4.2 Computing relative uncertainties

The complete algorithm for computing the relative uncertainty between any two nodes
in an arbitrary network is implemented by the Prolog procedure

\[
\text{relUnc(+FromNode, +ToNode, +IgnoreNodes, -Unc, -Traf, +Saving)}
\]
It takes the labels of two nodes, i.e. the names of objects instantiated in the workspace model, and a list of nodes to be ignored. It returns the relative uncertainty and the transform between the FromNode and ToNode, described in the FromNode frame. When determining the uncertainty, all paths through nodes that are in the set given by IgnoreNodes (which may be empty) are ignored. This amounts to excluding the information contained in specific parts of the network and is done when unmuting parts, as described in section 4.1.4. The Saving flag controls whether the recursive network description assembled during uncertainty computation is saved or discarded. If it is saved, then the next call to relUnc with the same arguments retrieves the saved data.

3.4.3 Observation integration

Recall that the integration of an observation into a network of relations requires changes in transformations such that the new network is consistent, and the changes are weighted by the information on the relations.

We use a recursive update procedure that is an extension of the one outlined at the end of section 3.1.4. For integrating a new observation between a and b, we use the structural information gained by the reduction of the network: The relation \( aR_b^{obs} \) representing the observation and the relation \( aR_b^{red} \) representing the reduced network are made consistent by making small changes to their transforms, weighted with their information. The change to \( aR_b^{red} \) is recursively propagated to its sub-relations, until the original relations are updated (The process may be stopped if the propagated change is smaller than a given bound).

We now state the algorithm: Given the description of a reduced path set \( ^aP_b \) with \( T_{Net}, \Lambda_{Net} \) and a new observation of the relation between a and b, with \( T_{Obs}, \Lambda_{Obs} \), the description first is extended to include the new observation, putting it in a series with the existing network.

\[
((a, a), \text{ser}( ((a, b), \text{Desc}_{net}, T_{net}, \text{Unc}_{net}), ((b, a), \text{inv simple}, \text{inv } T_{obs}, \text{Unc}_{obs})), T_{net} * \text{inv } T_{obs}, \text{zeros})
\]

While the actual transform of this relation is \( T_{Net}T_{Obs}^{-1} \), its nominal value is the identity transform. This inconsistency is eliminated by a recursive algorithm, implemented in the Prolog procedure

\[
\text{updateNetDesc(+Desc,+Traf,-NewDesc)}
\]

with the arguments:

\( \text{Desc: A description of a relation (x,y, Expansion, } \lambda_{x}, \lambda_{y}) \)
3.4. Realization of network functions

Traf: Its nominal transform \( T' \)

NewDesc: An internally consistent description \((x, y, \text{Expansion}^*, zT_y^*, zA_y)\) of the same relation: Its transform equals the nominal transform, and any differences to the original transform have been propagated to its components.

Algorithm: Take the given description apart and apply the algorithm to its components. Assemble the returned updated descriptions and return them to the caller. If the given description matches

a serial connection of relations, then distribute the difference between the actual and the nominal transform over the two relations, weighted by their information. This gives the nominal transforms of the two relations. Apply the algorithm to each of them.

a parallel connection of relations, then apply the algorithm to each of them.

a network of relations, then, for each of the relations of the network, compute the updated transform (see below) and apply the algorithm.

an inverse of a relation, then apply the algorithm to the inverse relation, using the inverse of the nominal transform.

a simple relation, then update the corresponding relation in the workspace model, using the nominal transform.

It remains to be shown how the updated transforms of an irreducible network are computed. Its nominal transform relates the cut-nodes which separate it from the rest of the network. We reduce the update problem to the problem of observation integration by viewing the nominal transform as an observation with virtually infinite information. This forces any inconsistencies to be absorbed by the other relations. In finite element theory, the corresponding approach (called "penalty approach") is used to take boundary conditions into account.

For exactly one observation \( z_{ij} \), having information \( \gamma_{ij} \), the right hand side of the update equation (3.29) becomes

\[
M_T z = [0, \ldots, 0, - \gamma_{ij} z_{ij}, 0, \ldots, 0, \gamma_{ij} z_{ij}, 0, \ldots, 0]^T
\]

Deleting redundant rows and columns of the IIM \( \Phi = M(\gamma + \gamma_z)M^T \) and inverting the resulting DIM, the change in node locations is obtained as

\[
\hat{x} = [\Phi^T_i]^{-1} \gamma_{ij} z_{ij}
\]

from which the changes \( \hat{x} \) in transforms are easily computed.
Summary

When planning the execution of an acting agent, its applicability will be judged by computing the relative uncertainty (Section 3.4.1) of participating objects and comparing it to the maximum uncertainty required for the agent (Section 3.2). If additional sensing is required, the uncertainty between the selected sensor and feature is determined and the feasibility of the sensing operation is verified. The description of this relation is saved and used when integrating the observation information (Section 3.4.3).

Modeling the effect of the acting agent on relations is discussed in the next chapter.
Chapter 4

MODELING AND PLANNING

The previous chapter introduced procedures for representing and manipulating uncertainties. This chapter shows their integration into a robot workspace modeling system and their use in planning acting and sensing operations. The first section shows how the influence of acting and sensing operations on transforms and uncertainties is modeled. The data for these operation models must be provided by the workspace model. The workspace model is expressed in terms of a “knowledge model”, a formalism for defining the structure of a knowledge base. The workspace model uses a hierarchy of classes to represent objects, their properties and geometric and logical relations. It includes potential relations, used when planning actions, and actually established relations, resulting from the execution of actions. Simple planning procedures are presented, showing the application of the workspace model for planning acting and sensing operations.

4.1 OPERATION MODELS

When an object (or a set of objects) is grasped, moved or observed, then its estimated position and uncertainty change. In order to have a basis for planning, we wish to model these changes. This will be done by creating, changing and deleting relations between objects.

The basic idea is to represent all uncertain geometric relations by homogeneous transforms and their associated uncertainties. This approach differs from the representation by general, parameterized functions with a probability distribution on the parameter vector [DW88]. It has the advantage of being a uniform representation, allowing to reason about uncertainties and partial information in a common framework. The loss of generality does not matter: If the relation of one object with respect to another is uncertain along e.g. a plane, a straight line, a circle, a spiral etc ... it is easily described by the uncertainty matrix of a single UT. Geometrically more complicated uncertainties can be assembled from a sequence of several simple UTs.

The actual workspace model represents one relation between objects by a sequence of three UTs. In this section, in order to simplify the presentation, we consider relations as single UTs.
The robot system interacts with its environment by acting and sensing. For our purpose we classify operations as

**Manipulation**, where the state of the parts in the workspace is changed. Manipulation may use sensors, e.g. for controlling or terminating a move.

**Observation**, where the state of the parts in workspace is not changed. Observation may use actuators, e.g. to position a sensor.

Usually, manipulations are further classified as *gross motion* or *fine motion*: A gross move has to *avoid* contact between objects during motion; a fine motion has to *create* contact between objects. We shall describe manipulations in terms of *mating*, *unmating* and *moving* operations, which we define as follows:

**Mating** involves a motion that creates a physical joint between objects, including grippers. Therefore mating includes both the grasping and the assembly of parts. The physical joint created is *fixed*, e.g. when grasping an object or doing a force fit, or *loose*, e.g. when placing an object on the table.

**Unmating** involves a motion that removes a physical joint between objects, undoing a mate operation. Some joints require a force greater than a certain threshold, others cannot be disassembled without damaging the objects.

**Moving** changes the joint transformations of an actuator. Any objects affixed to the actuator are also moved. Moving may be a part of mating or unmating.

Ideally, all manipulations could be modeled by the motion of objects and the forces they exert on one another. In reality, this is neither possible nor necessary. We shall introduce a number of restrictions under which modeling is simplified. In addition, the effects of gravity and the slack of joints will not be modeled, such that parts may e.g. not be dropped without support.

**Modeling vs. simulating**

A workspace model is used in two different situations (Fig. 4.1). One model represents the system's beliefs about its workspace. This model will be called the *planner model*. It is used to plan actions and interpret observations.

Another model replaces the actual workspace by a computer simulation. This model will be called the *simulator model*. It may be used to introduce errors, such that the simulation (like the real world) does not match the planner model. Then the effect of pose errors and the robot system's ability to correct them can be studied.

**4.1.1 Mating: Joint kinematics**

In both grasping and assembling, two objects make contact and comply to one another's pose until they are assumed to be joined. They shall be called *top* and *bottom*
4.1. Operation models

Figure 4.1: Modeling and simulating.

Figure 4.2: Examples for mating. The block at left is free to slide on the supporting plane. The block to the right is fixed to its support, and the large cylinder is free to slide in one direction and to rotate about its axis.
object. Since grasping an object is considered a special case of mating, "top object" may also denote a gripper. Consider Fig. 4.2: To the left, a block is shown prior to being grasped. When closing the gripper, the block will be shoved in the y-direction. If the error in gripper orientation is small, the block's position in the x-direction will remain unchanged. A new relation between the gripper and block is created. Its uncertainty in x-direction equals their relative uncertainty prior to gripping, and its uncertainty in y-direction reflects the gripper's tolerances.

To the right, a cylindrical peg is has been inserted in a larger cylinder. This cylinder rests on a fixture allowing rotation and sliding in one direction. After mating, the kinematic loop is

- overconstrained with respect to translation along the y-axis and rotation about the x-axis: No relation of the loop can be changed without deforming any objects.
- underconstrained with respect to rotation about the z-axis: The large cylinder is free to rotate relative to the small one.
- just constrained with respect to translation in the x-direction: The position of each object in the loop is completely determined, but not overconstrained.\(^1\)

In general, there may be several parts in the kinematic chain between the actuator and the top object, as well as between the bottom object and the table. If the mate operation is successful, it will know that it has achieved the goal relation between the top and bottom object. Since the objects have moved with respect to one another, their relative uncertainties will have been (partly) invalidated. What are the changes in the relative poses and uncertainties of the objects within the kinematic loop created by the mate operation?

The object poses are given by the transforms of the joints in the loop. Each joint transform in turn is determined by

- constraints imposed by the joint itself: Every correctly established joint limits the relative freedom of motion of the joined objects in a known way.
- constraints imposed by the kinematic loop: The relative freedom of motion of two joined objects is limited by the freedom of motion of all the other joints around the loop, and the condition that the loop be closed.
- friction and stiction in those directions in which the loop is underconstrained: At least one part of the loop is free to move with respect to the other. The final poses are determined by the motions of objects during mating.

Ideally, only the actuator and the grasped parts move during mating, following the planned trajectory. In reality, a search for the correct mating pose may be required, and both the top and bottom parts may move to comply to one another's pose. Compliance may occur in one or several directions, while in the orthogonal directions the

\(^1\)Such lines of thought are part of the study of kinestatics, i.e. the kinematics and statics of mechanisms [Phi84].
objects may remain free move relative to one another. Compliance takes place when either the manipulator moves the top object to a predetermined pose and the (it is to be hoped) flexibly supported bottom object is forced to comply, or when the bottom object is rigidly supported and the top object complies. In typical situations of robotic assembly, the first case occurs when grasping objects, with chamfered grippers reducing uncertainty and objects being loosely supported by pallets or feeders. The second case may occur when joining objects. A combination of both cases may also occur.

Restrictions on the modeling of mating operations

In general, we assume that both the top and bottom objects comply. The bottom object is pushed in one direction while the top object complies in another or in the same direction. When there is friction, the exact motion of a pushed object usually cannot be determined. For example, consider an object on a flat surface being pushed by a point contact. If the distribution of support forces of the object is indeterminate, then the motion of the object is indeterminate [Mas85]. Only its direction of rotation may be found. The scope of applicability of a quasi-static analysis of pushing (i.e. neglecting inertial forces) is derived in [Mas86].

A general simulation of object motion during mating therefore is impossible. If the mating operation involves a complicated search for the right mating position, during which the bottom object is moved, then little about final positions can be inferred from a priori positions. One way to get around this problem is to support the bottom object rigidly, forcing all compliance to take place in the actuator. Since we are interested in modeling realistic assembly situations, we shall allow objects to be supported loosely. This forces us to constrain the generality of motions during mating, making a reasonable compromise between model simplicity and applicability. If the following restrictions are satisfied, the joints in the kinematic loop will comply to accommodate the constraints introduced by the new joint, and the final poses can be determined.

1. The joints can be described by a set of standard joints.
2. The relative motion between the two objects being mated is a sequence of linear moves and rotations.
3. The joints of the kinematic loop comply passively or actively and errors and displacements due to compliance are small.
4. The joint tolerances after mating are small.

Description of physical joints

We model the compliant motion of objects by a procedure analogous to the integration of UTs. Depending on the resulting changes in object poses, the uncertainty of each relation is changed. We first show how the possible motion of joints is described and how this description is used to determine the displacements and uncertainties after mating.
Figure 4.3: The six lower joints. The kinematic properties of a joint or combination of joints can be expressed in terms of these six simple or lower joints [Phi84].

Figure 4.4: Contact geometries in assembly: Planar contact, prismatic joint and rotating joint. The corresponding joint constraint matrices have the diagonals \([1,1,0,0,0,1]\), \([0,1,0,0,0,0]\) and \([0,0,0,0,1]\). The associated coordinate frames are assumed to be parallel to the frame shown.
Consider two objects being mated, called top and bottom part and labeled T and B. The freedom of motion in the relation from T to B can be described by the joint tolerances, which are part of the geometric model. We wish to describe this freedom of motion in terms of uncertainties, such that the information can be used in the uncertain geometric model. We define a special coordinate frame, the joint frame, such that the linear and/or angular degrees of freedom of the joint are along its axes (Fig. 4.3 and 4.4).

**Definition 3** The maximum joint uncertainty $\mathbf{T}_j$ expresses the tolerances and the degrees of freedom of the joint between frames $i$ and $j$ in terms of uncertainties, in a frame suitably aligned with the joint.

For any successfully established joint, $\mathbf{T}_j \succeq \mathbf{A}_j$ must hold. $\mathbf{T}_j$ is diagonal, and its entries are derived from the joint tolerances, using a simple statistical model, similar to [MB88]: For each direction, the variance of the error distribution is chosen such that the error lies within the given tolerance interval $[-t_i, t_i]$ with some confidence.

For a univariate normally distributed random variable with variance $\sigma$, the probability that its value lies within $\pm 3\sigma$ of its mean is 0.998, i.e. virtually one. Taking into account that the diagonal elements of a covariance matrix correspond to the squared variances, the maximum joint uncertainty for each direction should therefore be chosen as $(t_i/3)^2$ to $(t_i/4)^2$ of the corresponding joint tolerance.

For our multivariate distribution, the probability that tolerances are not exceeded in all directions is bounded by the Bonferroni inequality [KJ82], which states, for $n$ random events $A_i$ with probability $\Pr[A_i]$, that

$$1 - \sum_{i=1}^{n} \Pr[A_i] \leq \Pr\left[\bigcap_{i=1}^{n} A_i\right]$$

If $A_i$ is the event that the joint is within its tolerance in direction $i$, then for $3\sigma$ tolerances the left hand side is

$$1 - 6 \cdot 0.002 \leq \Pr\left[\bigcap_{i=1}^{6} A_i\right]$$

which still is close enough to 1.

For directions in which the joint is free to move, the tolerance is infinite. In order to avoid numerical difficulties, we set the maximum joint uncertainty to a large but finite number.

The maximum joint uncertainty completes the description of a geometric relation. For ease of expression, we define:

**Definition 4** The relation $\mathbf{R}_j = (\mathbf{T}_j, \mathbf{A}_j, \mathbf{T}_j)$ consists of the homogeneous transform, its uncertainty and the maximum joint uncertainty relating frames $i$ and $j$. A sequence of compounded relations is denoted in the same way as a sequence of transforms, e.g. by $\mathbf{R}_j \mathbf{R}_k$.

In addition, we define the joint constraint matrix $\mathbf{C}_j$, which shall be used later.
Figure 4.5: Position update. The pose of the top object T is completely known. During mating, the bottom object B may have complied by sliding on the supporting plane. The position of B, i.e. the relation $^{R}\text{B}_{W}$ must be updated in the directions of compliance.

Definition 5 The diagonal joint constraint matrix $C = [c_{ij}]$ associated with the maximum joint uncertainty $\Gamma = \gamma_{ii}$ is obtained by setting $c_{ii} = 0$ for $\gamma_{ii} \ll k$, and $c_{ii} = 1$ for $\gamma_{ii} \gg k$.

The number $k$ is joint and axis dependent. It should be chosen to be the geometric average between typical maximum and minimum uncertainties.

4.1.2 Mating: Pose update

The procedures for updating positions and uncertainties after a mating operation will be derived using the situation shown in Fig. 4.5. The procedures are used in both the planner model and the simulator model.

The planner model uses the expected pose of B and its nominal relation to T to compute the goal pose of T as $^{W}\text{T}_{T} = (^{T}\text{T}_{B}^{B}\text{T}_{W})^{-1}$. If it can be placed there successfully, then the expected pose of B must have been correct to within errors absorbed by chamfering. Although the position of B may have changed, this cannot be detected, since the loop of expected transforms is consistent, i.e. $^{W}\text{T}_{T}^{T}\text{T}_{B}^{B}\text{T}_{W} = I$. Therefore, no pose update is required.

But if the manipulator had to perform a search or comply to reach the mating position, or if somebody is interested in simulating the effect of pose errors, then the final transform $^{W}\hat{\text{T}}_{T}$ will not equal the expected $^{W}\text{T}_{T}$. The other two transforms must be changed such that $^{W}\hat{\text{T}}_{T}^{T}\hat{\text{T}}_{B}^{B}\hat{\text{T}}_{W} = I$.

In other words, the relations assumed to have held before mating and the relations known to have been enforced by the mating operation must be integrated. For a loop of transforms $\prod_{k=1}^{n} T_{k} \neq I$, small changes to the $T_{k}$ must be found to achieve consistency.
4.1. Operation models

Since the maximal joint uncertainties provide information about the transforms, just as observations would, we use the procedure for making a loop of observations consistent (§3.14). Let $\mathbf{d}$ be the differential motion vector describing the total loop error. Let $\Gamma_i, i = 1, \ldots, n$ be the maximum joint uncertainties of loop joints, expressed in the same coordinate system as $\mathbf{d}$. Then the change of joint $i$ is

$$\mathbf{d}_i = \Gamma_i (\sum_{k=1}^{n} \Gamma_k)^{-1} \mathbf{d}$$

If the loop is underconstrained in a specific direction, the above procedure will distribute the changes in that direction over the joints which are free to move. If the loop is overconstrained in some direction, then its stiffness in that direction is large and $\sum_{k=1}^{n} \Gamma_k$ is close to singular. If the error is not zero, then some parts must have been deformed.

4.1.3 Mating: Uncertainty update:

As seen in the introductory example, the uncertainty of a joint depends on the uncertainty prior to mating, and on the maximum joint uncertainty of the joint itself and of the other joints in the kinematic loop. The influences of these uncertainties in a particular direction depends on how the loop is constrained.

Consider one of the six directions of a particular joint. All uncertainties are assumed to be represented in its joint frame. One of the following cases occurs:

1. If the joint itself constrains motion, then the uncertainty after mating equals the maximum joint uncertainty.
2. If the motion of the joint is constrained by the loop, then the uncertainty of the joint itself can be set arbitrarily large. This is because the joint will have moved during mating and its uncertainty is completely determined by the uncertainties around the loop.
3. If the loop is underconstrained, then the joint may or may not have been moved. If it has not moved, then its uncertainty equals its uncertainty prior to mating. The worst case assumption would be to assume all directions of all joints free to move to have moved. This implies discarding all information in these directions and is unrealistically pessimistic. The assumption made here is that any direction of a joint not forced to move has not moved, hoping that this is not unrealistically optimistic.

What is the resulting uncertainty, if the prior uncertainty in one direction (case 3) needs to be combined with the maximum joint uncertainty in another direction (case 1)? In the joint frame, the prior uncertainty is described by an arbitrary covariance matrix, while the maximum joint uncertainty is diagonal. The combined uncertainty should have zero covariance between the two directions, since moving the joint in one direction destroys the correlation with the other directions. The marginal density should remain the same in the direction in which the uncertainty is not changed.
The variance of the marginal density is just the corresponding diagonal entry of the covariance matrix. Therefore, we assemble a diagonal posterior uncertainty matrix with entries taken from the diagonal of the maximum joint uncertainty (case 1), the prior uncertainty (case 3), or set arbitrarily large (case 2).

It remains to be shown how the above cases can be differentiated. We use the joint constraint matrix defined earlier. It has the same structure as the maximum joint uncertainty and can be manipulated essentially like an uncertainty matrix. We replace the maximum joint uncertainty in (4.1) by the joint constraint matrix, which gives

$$d_i = C_i \left( \sum_{k=1}^{n} C_k \right)^{-1} d$$

(4.2)

where the $C_i$ are the joint constraint matrices of the joints forming the kinematic loop, transformed to some arbitrary but common coordinate frame. The equation is an idealized version of (4.1), with uncertainties being reduced to either "small" or "large". The update cases can now be differentiated by considering the effect of an arbitrary loop error $d$ on each joint: For joint $i$, we define $e = [e_j]$ to be equal to $d_i$ and $L = [l_{jk}]$ to be equal to the expression $C_i \left( \sum_{k=1}^{n} C_k \right)^{-1}$, transformed to frame $i$. Then (4.2) can be written as

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \end{bmatrix} = L \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{bmatrix}$$

and we see that:

1. If $l_{ik} = 0, k = 1, \ldots, 6$, then $e_j = 0$, regardless of $d$. Direction $j$ of the joint transform cannot be changed by the rest of the loop, i.e. the joint constrains itself, and its uncertainty is set to the maximum joint uncertainty. This can also be seen the other way round: If the joint is constrained in direction $j$, then $[c_{ij}] = 0$. Since $C_i$ is diagonal in frame $i$, its entire row $j$ is zero and therefore also row $j$ of $L$.

2. If $l_{jj} \approx 1$, then with $d_k = 0$ for $k \neq j$ we have $e_j \approx d_j$. The entire loop error in direction $j$ is absorbed by joint $i$, and its uncertainty is set arbitrarily large. The relative position and the uncertainty of direction $j$ of joint $i$ is determined by the loop.

3. If $l_{jj} \approx 0.5$, then only part of the loop error in direction $j$ is absorbed by joint $i$. The same must hold for at least one other joint $h$. The two parts of the loop separated by joints $i$ and $h$ are free to move with respect to one another. The loop is underconstrained. The displacements in joints $i$ and $h$ are correlated, which we choose not to model. The uncertainties in both joints are set to the corresponding a priori uncertainties. If a search in direction $j$ is known to have taken place, the uncertainty can be set according to the magnitude of the searching motion.
If $\sum_{k=1}^{n} C_k$ is singular, the loop is overconstrained in at least one direction. We may either restrict computations to the other directions or avoid numerical problems by adding a small value to the diagonal entries of the $C_k$ which are zero. This allows to distinguish the above cases without further complications.

The direction of singularity indicates in which direction active compliance is required. This information could be used by an agent executing the mate operation.

The displacement of an object affects not only information on the kinematic loop, but also any observations of the object. Their update can be determined in the same way, by examining the potential relative motion between the object and the opposite nodes of the observations.

Summary

When mating two objects, the following changes to the workspace model are made:

- A new joint instance is created
- Transforms of joints around the new kinematic loop are made consistent
- Uncertainties of joints around the new kinematic loop are updated
- Uncertainties of observations to parts in the loop are updated

They are implemented in the Prolog procedure

```prolog
create_phys_joint( +Top, +Bot, +JointId)
```

where Top and Bot denote part instances and JointId gives the id of the joint instance to be created.

4.1.4 Unmating

Unmating comprises disassembly, which occurs seldom, and ungrasping, which happens very often. If the operation does not move the bottom object, the bottom object's uncertainty is also unchanged. This assumption is reasonable for both kinds of unmating operations: When ungrasping, the bottom object will not move if it is stably supported and does not stick to the gripper fingers. When disassembling, either the parts are joined loosely, or the bottom part is firmly held by a fixture or another manipulator.

The main problem when unmating is what to do with the information in physical joints and observations being deleted.

Notation  Recall that a physical joint may be fixed or loose. In addition, relations representing observations shall be labeled as fragile. We define the following sets of nodes or relations:
Figure 4.6: Node and relation sets involved when unmating.

- Affixed nodes: $FN_{N,I}$ is the set of nodes for which there exists a path to node $N$, such that all relations of the path are fixed, and no node in the set $I$ is contained in the path.

- Mated nodes: $MN_{N,I}$ is the set of nodes for which there exists a path to node $N$, such that all relations of the path are nonfragile (i.e. fixed or loose), and no node in the set $I$ is contained in the path.

- $FR_{N,I}$ is the set of relations of which exactly one node is in $FN_{N,I}$. It follows that all relations in $FR_{N,I}$ are loose or fragile.

- $MR_{N,I}$ is the set of relations of which exactly one node is in $MN_{N,I}$. It follows that all relations in $MR_{N,I}$ are fragile.

- $J(A, B)$ is the set of relations having one node in $A$ and the other in $B$, where $A$ and $B$ are node sets. $A$ and $B$ may also be single nodes.

This allows to express the different sets of relations involved in unmating (Fig. 4.6). The nodes being separated are labeled $T$ and $B$. The set of parts affixed to the top part, excluding the bottom part and the gripper, is $FN_{T,(B,G)}$. The set of parts mated to the bottom part, excluding the top part and the table, is $MN_{B,(T,W)}$.

$TBD$: Direct relations between $T$ and $B$: $TBD = J(T,B)$

$TBI$: Indirect relations between $T$ and $B$:
\[ TBI = J(FN_{T,(B,G)}, MN_{B,(T,W)}) \]
Note that $TBD \subseteq TBI$.

$TO$: Observations of parts affixed to $T$:
\[ TO = \{ x \mid x \in FR_{T,(B,G)} \land x \notin TBI \} \]

$BO$: Observations of parts mated to $B$:
\[ BO = \{ x \mid x \in MR_{B,(T,W)} \land x \notin TBI \} \]

We now show how these relations are affected by unmating $T$ and $B$: 
4.1. Operation models

Figure 4.7: Relations after unmating. To simplify the representation, the relations from the world to the gripper are represented by a single relation $W_R_G$.

Physical joint information

The relation representing the no longer existing physical joint is deleted. In general, all relations in $TBD$ are deleted. The update is simplified if they all first are merged into a single one. What is to happen with the information contained in the corresponding relation? The situation will appear as in Fig. 4.7. As $A$ is assumed not to have moved, the information of $^G R_A$ is still valid. Simply discarding the information along with the relation is not appropriate, since $W_R_G^G R_A$ contains information about the pose of $A$ with respect to $W$. There are two possibilities to keep that information: It may be propagated into the other relations of the loop, or an observation containing the information may be created. The second approach is to be preferred:

Taking the first approach, propagating the information of $^G R_A$ via $^A R_B$ and $W_R_G$ to $^B R_W$ and integrating it with the existing information of $^B R_W$ gives the updated $^B \Lambda_W$ (and similarly $^A \Lambda_B$):

$$
^B \Lambda_W = (^B \Lambda_W^{-1} + (^W \Lambda_G + ^G ^B \Lambda_A + ^A ^B \Lambda_B)^{-1})^{-1}
$$

$$
^A \Lambda_B = (^A \Lambda_B^{-1} + (^A ^B \Lambda_W + ^W \Lambda_G + ^G ^A \Lambda_A)^{-1})^{-1}
$$

But now the updates $^B \Lambda_W$ and $^A \Lambda_B$ are no longer independent, and their cross uncertainty is no longer zero. Ignoring it would amount to the same as ignoring the information contained in $W_R_G^G R_A$. Not ignoring it necessitates keeping track of all cross uncertainties between relations and taking them into account in all computations.

Taking the second approach, a relation $W_R_A$ representing an observation is created, replacing $W_R_G^G R_A$. The network contains the same information as with the first approach, with the advantage that cross uncertainties remain zero. Whenever necessary, the correct total information from $A$ to $W$ can be determined from the network of
relations. If, later in the assembly process, parts get moved, the information will be updated or discarded correctly by the procedures for updating observed relations.

Observations of grasped parts

Consider a part or assembly that has been grasped and will be moved. It will be unmated from its support. There may exist observations of the grasped part(s), described by the relations in $T_O$. They clearly become invalid as soon as the parts are moved along with the gripper. Simply discarding the observations may amount to discarding important information about the relative pose of the grasped parts.

The information is retained by the following procedure: All observations of grasped parts are replaced by observations joining the parts with the gripper. The uncertainties of these new observations are obtained by adding the uncertainty between the gripper and the respective sensors to the uncertainties of the deleted observations. The information of the new observations will be unaffected by gripper motion, since there is no relative motion between the gripper and the parts affixed to it.

Observations of remaining parts

There may also be observations that join $T$ and $B$ indirectly, described by the relations in $T_B I$. For example (see Fig. 4.7), there may be observations joining the gripper and parts $A$ or $B$. When ungrasping, an observation to $B$ may be propagated either to the world frame or to $A$ in order to retain some of its information. The choice depends on the other uncertainties: If $W_A B > W_A G$, then propagation to $A$ sacrifices more information than propagation to $W$. We nevertheless choose always to propagate information to the bottom part (A in this case). Otherwise, if a relation from $B$ to $W$ had been created, then when picking up $B$ later, its information would have to be propagated from $W$ back to the gripper (see the preceding paragraph).

Summary

When unmating two objects, the following changes to the workspace model are made:

- The relations in $T_B D$ are deleted and their information is collected.
- Relations in $T_B I$ are propagated to $B$, creating relations from $B$ to nodes in $M_N B,\{T,W\}$.
- The relative information between $B$ and $W$ is determined, ignoring all paths which contain nodes in $F_N T,\{B,G\}$. A new observation from $W$ to $T$ is created, incorporating this information and the information of $T_B D$.
- The relative information between $T$ and $W$ is determined, ignoring all paths which contain nodes in $M_N B,\{T,W\}$. A new observation from $W$ to $B$ is created, incorporating this information and the information of $T_B D$. 
• Relations in $TO$ are propagated to the gripper $G$, creating relations from the gripper to nodes in $FN_{T,\{B,G\}}$.

Some of the above steps may be omitted if $T = G$ or if $B = W$. All updates are made by the Prolog procedure

```
delete_phys_joint( +Top, +Bot, +JointInst)
```

where Top and Bot denote part instances and JointInst gives the instance of the joint instance to be deleted.

### 4.1.5 Moving

A gross move of a grasped part $P$ changes only $\text{wR}_G$ and leaves all other relations, in particular $\text{gR}_P$ unchanged. The relations in $\text{FR}_{PG}$ become invalid, but if the preceding unmate operation was completed successfully, then $\text{FR}_{PG} = \emptyset$. After the move, the uncertainty in $\text{wR}_P$ can be found from $\text{wR}_G\text{gR}_P$. As stated above, it needs to be determined only after ungrasping the object, when the relation $\text{gR}_P$ and its uncertainty are discarded. They corresponding Prolog procedure is

```
change_phys_joint(+Top,+Bot,+JointInst,+NewTraf,+NewUnc)
```

where Top, Bot and JointInst specify instances and NewTraf, NewUnc give the new joint transform and uncertainty.

### 4.1.6 Example of manipulation

The following example shows the addition and deletion of joints during the manipulation of parts: The initial state is that of Fig. 4.8 a): Part LC is affixed to SB, which lies on the table. Similarly, SC is stacked on LB. The poses of LC and LB have been observed by a camera (C).

When grasping SB with the gripper G, a new relation $\text{gR}_{SB}$ is created (Fig. 4.8 b). The uncertainties and transforms of $\text{wR}_A$, $\text{aR}_G$, $\text{gR}_{SB}$, $\text{sBR}_W$ are updated, as well as the uncertainty of $\text{gR}_{LC}$. Picking up SB, i.e. unmating it from the table, deletes $\text{sBR}_W$ and creates an observation $\text{wR}_{SB}$ containing the information lost otherwise (Fig. 4.8 c). $\text{wR}_{SB}$ in turn is deleted and its information propagated to $\text{gR}_{SB}$. The observation information in $\text{cR}_{LC}$ is propagated to $\text{gR}_{LC}$.

Mating SB to LB creates $\text{sBR}_{LB}$ and updates the new loop. Finally, ungrasping SB deletes $\text{gR}_{SB}$ and saves its information in $\text{wR}_{SB}$. $\text{gR}_{LC}$ is replaced by $\text{sBR}_{LC}$. The final state is shown in Fig. 4.8 d).
Figure 4.8: Joint changes during manipulation. A: Arm, G: Gripper, C: Camera, W: World, SB, LC, LB, SC: Parts. Fragile, loose and fixed relations are denoted by thin, medium and thick arrows.
4.1. Operation models

4.1.7 Taking observations

A sensing agent is given the name of a feature to be observed and possibly the directions in which pose information is required. If visibility and focus constraints are satisfied, the agent can be applied. Raw sensor data is acquired and processed by the sensing agent and expressed as an uncertain geometric relation between the sensor and the feature. The new relation must be tested whether it is consistent with expectations. If it is accepted, it must be integrated into the existing network of relations. Procedures for the consistent integration of observations into loops or networks of relations have been given in sections 3.1.4 and 3.3.3. The decision on the acceptance of observations is based on the difference of the estimates which are to be merged. For two observations \( z_0 \) and \( z_1 \) with covariance matrices \( \Lambda_0 \) and \( \Lambda_1 \), a measure of difference is the Mahalanobis distance [HS90]

\[
d_M = \frac{1}{2}(z_0 - z_1)^T(\Lambda_0 + \Lambda_1)^{-1}(z_0 - z_1)
\]

An upper bound on \( d_M \) states that the differences between observations, weighted with their relative uncertainty, may not exceed a certain threshold. Larger uncertainties \( \Lambda_i \) allow larger differences to be tolerated. In [DW88], it is argued that \( d_M \leq 1 \) is required for a bayesian consensus.

Sensor and feature uncertainties

The information acquired by most sensors is only partial: For example, a vision system cannot determine the location of a feature along the viewing direction of the camera, if there is no a priori information about the feature size. Furthermore, the pose of a feature only partially constrains the pose of its object: If the pose of a straight line or a plane of an object is known, the object still is free to move along and rotate about the line or within the plane. We therefore distinguish the actual feature, whose relation to the object is completely determined, and the observed feature. The observed feature has the same pose as the actual feature, except for those directions in which the feature provides no information. We define the feature uncertainty matrix to represent this lack of information. It is feature specific. If its frame of description is suitably aligned with the feature, it will be diagonal with entries (ideally) being zero or infinite.

The sensing uncertainty has both feature specific and sensor specific origins. Consider Fig. 4.9 for the case of a line being observed by a camera. If the length of the line is not known, then the camera provides no information about the translation of the line along \( z_S \) and along the line \( \overline{SF} \) and about its rotation about \( y_F \). The accuracy of the sensor can be expressed as angular uncertainty in frame \( S \), or, magnified by the distance \( \overline{SF} \), as linear uncertainty in frame \( F \).

We want to decouple this kind of reasoning, which is specific to a particular sensor/feature configuration, from the general workspace model. We therefore require the sensing agents to handle the configuration specific aspects and to return a single uncertainty matrix \( S\Lambda_F \), describing the complete uncertainty between feature and sensor in the frame of the observed feature. For transforming the different uncertainties
Figure 4.9: Observation uncertainties. When observing a line with a camera, there is no information about the distance in the viewing direction and the relation between the observed line and the actual line.

into that common frame, the transform $S^T_F$ is required. But since the observation of $S^T_F$ is incomplete, it must be complemented by the a priori estimate of $S^T_F$. In our example, an estimate for the distance $S^F_F$ is needed, since it is not provided by the observation but does have a nonnegligible effect on the transformation of uncertainties. The consistent integration of observations into the existing network of UTs also must use this complemented $S^T_F$ in order to keep the loop error small, such that the assumptions for linearity of the problem hold.

Summary

When taking an observation, the sensing agent

- complements the observation data with the data from the expected relation,
- determines the complete observation uncertainty, and
- checks the acceptability of the observation.

Then the workspace modeling system

- creates an observation joint containing the complemented observation and its uncertainty, and
- integrates it into the network, updating transforms to achieve consistency.

The whole is implemented by the Prolog procedure

create_obs_joint(+Top,+Bot,+JointId,+ObsTraf,+ObsUnc)
where Top and Bot denote part instances and JointId gives the joint id of the observation. ObsTraf, ObsUnc are the transform and uncertainty from the sensor to the observed feature.

4.2 KNOWLEDGE MODEL

Most of ARGUS is realized in the Prolog language. Good programming style includes hiding data structures and limiting access to data to a clearly defined set of routines which guarantee data consistency. While the Prolog language includes procedures for storing and retrieving user data easily and efficiently, it does not provide mechanisms for data encapsulation or modularization as e.g. Modula-2 or Ada do. It also lacks a typing concept, such that syntactically correct clauses may be incorrect in the context of a specific application. These circumstances have lead to typing concepts and knowledge models for implementing databases in Prolog: Type-inherent consistency rules describe the structure and the arguments of legal predicates, and type-external consistency rules define the relations that must hold between different predicates. A knowledge model is defined as a formal language that allows to describe

- entities and their relations.
- legal operations on these entities
- rules to be followed when operating on the entities, enforcing consistency between entities

ARGUS uses such a knowledge model to implement the workspace model. This is particularly useful when experimenting with the system: Model changes are likely to be entered by the user and therefore susceptible to errors. Type and consistency checks can detect these errors.

The knowledge model is derived from the "Prolog Knowledge Model" (PKM) [Est89] and is now described.

4.2.1 Knowledge model definition

The entities represented by the knowledge model are defined by a set of classes, their properties and relations. The class definitions are processed by Prolog and provide a framework for updating and retrieving instance data. A class definition may include type-inherent consistency rules and rules for maintaining model consistency. It has the syntax

'class' Class-name
  'attributes' : Attribute-Value-Definition
  'identifier' : Attribute-name
  'is.a' : Parent-Classname
  'rules' : Rule-List .'}
Chapter 4. Modeling and planning

where

- **Attribute-Value-Definition** is a Prolog list whose entries have the form

  \[
  \text{Attribute-name: type: characteristics}
  \]

  The type of an attribute is either the name of another class or the name of a type checking Prolog procedure. The type checking procedure may also convert different representations into a common format. E.g. transforms may be specified by their $6 \times 1$ description vector or by their $4 \times 4$ homogeneous transform matrix. In both cases, the same internal representation will be obtained. The characteristics entry is optional and may contain any of the following keywords

  - **unique**: The value of the attribute must be unique for all objects of the same class.
  - **required**: The value of the attribute must be provided when creating a class instance.
  - **unchangeable**: The value of the attribute may be instantiated at most once.
  - **multivalued**: The attribute may have not just one value but a set of values.

  A violation of the characteristics is detected during update operations and causes an exception to be raised.

- **Attribute-name** following the 'identifier' keyword is the name of the attribute whose value uniquely defines an instantiation of the class.

- **Parent-Classname** is the name of the attribute whose value is the name of a parent class from which attributes and values will be inherited. The parent class definition must have been processed prior to the processing of the child class. Otherwise, an error will be signalled. This entry is optional.

- **Rule-List** specifies Prolog procedures to be executed by each of the update procedures. Its entries have the form

  \[
  \text{'on' update-op 'do' clause}
  \]

  where update-op is one of \{ 'insert', 'modify', 'delete' \} and clause is a Prolog clause. Its arguments must be attribute names of the class whose definition contains the rule. When executing an update operation, the corresponding clause is called with its arguments instantiated to the actual values of these attributes. For the 'insert' operation, the clause is called after its execution, for the 'delete' and 'modify' operations, before their execution. The 'rules' entry is optional.

Example: A class is defined by
4.2. Knowledge model

![Class Diagram]

Figure 4.10: Parent class and superclass relations. Classes B and C are child classes of class A. They inherit all attributes of A. Class S is a superclass, containing all instances of classes C and D.

```plaintext
4.2.2 Class hierarchies

The 'is_a' entry introduced above specifies that parttype is a parent class of partid class (see Fig. 4.10). In addition, we use the following terms:

Definition 6 If class X has parent class Y, then X is a child class of Y. X is a descendant class of Y, and all descendant classes of X are also descendant classes of Y. Y is an ancestor class of X, and all ancestor classes of Y are also ancestor classes of X.

The instance of a class has an instance of the parent class as a parent instance (or simply “parent”). It inherits all attributes and their values from the ancestor instances. If an attribute is defined for a class as well as for one or more of its ancestors, then the value in a descendant takes precedence over a value in an ancestor. The value of the ancestor is returned only if the value in the descendant is not instantiated.

For example, an instance of a peg-in-hole joint may have attributes describing instance specific geometric relations, and inherit attributes from a class instance describing generic peg-in-hole joints.
```
A second relation between classes is described by the concept of a superclass: It may be necessary to refer to instances of different classes collectively, without having to bother which class the instances actually belong to. For example, sensor observations and physical joints both relate objects and contain information about the geometry of the relations. When computing the transformation or the uncertainty between objects, it is irrelevant what class the relation belongs to. The algorithm processing the relations should not have to know the two classes and refer to them explicitly. Therefore, a superclass is defined to contain the two classes, whose instances then also are instances of the superclass (Fig. 4.10). When retrieving data from the instances of a superclass, all its member classes are searched. A superclass is declared by

'superclass' Superclass-name 'contains' class-list

where class-list is a list of class names.

4.2.3 Update and retrieval operations

Class instances can be created (inserted), modified and deleted, and their information may be retrieved. These operations have the syntax:

Operation-name Class-name 'with' Attribute-Value-Set

where Operation-name is one of { 'insert', 'modify', 'delete', 'retrieve' } and Attribute-Value-Set is an unordered set of attribute-value pairs. For each type of operation, Class-name must be the name of an existing class, and the following conditions on attributes and values must hold:

insert:

- For an attribute with the 'required' characteristic, the value must be given.
- For an attribute with the 'unique' characteristic, the value may not be the same as that of an existing instance of the same class.
- Attribute names must be the same as given in the class definition.
- Attribute values must fit their type as given in the class definition.
- If the type of an attribute is a class or superclass, then its value must match the name of an instantiation of that class or superclass.
- Multiple values may be given only for attributes with the 'multivalued' characteristic.
- The execution of consistency rules must succeed.
- A value must be instantiated, i.e. it may not be an uninstantiated Prolog variable.

Example: An instance of the previously defined class partid is created with

insert partid with [id : washer1, type : small_washer].
modify, delete: In addition to the conditions given for the 'insert' operation, the following conditions must be satisfied:

- The attribute specified as identifying instances (by the 'is_a' entry) must be given.
- The instance must exist.
- The values of attributes with the 'unchangeable' characteristic may not be changed.

Example: An instance is modified with

\[
\text{modify partid with [id : washer1, color : blue, placetraf : [15, 15, 0, 0] ].}
\]

retrieve:

- Class-name may also be the name of an existing superclass.
- If an attribute does not exist or is not instantiated, then the ancestors are searched. The attribute must exist in one of the ancestors, but need not be instantiated. If it is not instantiated, the value default is returned.

When retrieving, attribute values may be instantiated or not, as in ordinary Prolog procedure calls. If they are instantiated, the set of matching instances is constrained. If they are not, all matching instantiations can be found on backtracking. For example,

\[
\text{retrieve partid with [id : Id, color : blue, type : small_washer, placetraf : Pt ].}
\]

matches, upon backtracking, the ids and placetrafs of all instantiations of small blue washers.

When matching multivalued attributes, the following conventions are applied: Let \( R \) be the value of an attribute given in the retrieve request, let \( D \) be the corresponding value or set of values in the database, and let \( \text{var}(X) \) be a function that returns true if \( X \) is not instantiated. The rules for matching are: If \( \text{var}(R) \) then \( R \) is matched with \( D \). If \( \neg \text{var}(R) \lor R \in D \) or if \( \neg \text{var}(R) \lor R = D \), then the match succeeds. If \( R \) is a list of values, it must match \( D \) exactly, including the order of values.

Two additional procedures are provided for the convenience of the interactive user: The commands

'\text{dispcl } Class\_name \ '.

'\text{dispid } Instance\_name \ '.

list all instances of the class or superclass \( Class\_name \) and all non-default attribute-value pairs of the instance \( Instance\_name \).
4.3 WORKSPACE MODEL

The workspace model must on the one hand provide planners with expectations about the state of the workspace, and on the other hand provide procedures to model changes in the workspace as operations are executed.

Assembly is concerned with objects and their geometric relationships. The obvious modeling approach is to use a graph, with nodes corresponding to objects and arcs corresponding to relations. As objects enter and leave the workspace, nodes are created and deleted, and as objects are observed and moved around, arcs are created, modified or deleted.

4.3.1 Representation hierarchy

Blocks world planners consider a set of blocks and generate plans for putting block A onto block B. In the real world, however, A may be a M3×30 screw, and C also may be a M3×30 screw. The planner should take advantage of the fact that A and C are “the same”. Similarly, two assemblies which are “the same” may be assembled simultaneously, both consisting of “the same” parts. On the other hand, two parts that are “the same” may be used in a different way in the same assembly. E.g. the screw at the upper left and the screw at the lower left of the assembly. These different notions of sameness lead to a hierarchy of classes representing objects and their relations. For objects, its elements are defined as follows (Fig. 4.11):

- All objects with the same physical appearance share the same type. They appear the same to the various sensors, or differ only within a specified range. An example of a part type is “a M3×30 screw” or “a 28-inch bicycle wheel”. In the knowledge model, classes representing types have the suffix “type”.

- All objects of the same type that are used in the same way share the same identity (or id). “Used in the same way” means that they have the same function and position within identical assemblies. An example of a part id is “the upper left screw holding the cover” or “the front wheel of a swiss army bicycle”. In the knowledge model, classes representing objects with the same identity have the suffix “id”.

- Every object is represented by a unique instance. An example of a part instance is “the upper left screw holding the cover of the assembly with serial number 37886”. In the knowledge model, classes representing part instances have the suffix “inst”.

For the set of objects in Fig. 4.11, the different class instances might be:

\[\text{Note the two different uses of the term “instance”: In knowledge model terminology, it stands for an entity created from a class definition, while in workspace model terminology, it denotes a level of the hierarchy of the model.}\]
A similar hierarchy describes relations: A relation type describes a type of an observation or a physical joint. A relation id describes a relation between part ids, and a relation instance describes a relation between part instances.

- A relation type describes a type of an observation or a physical joint. Examples are `straight_edge`, `parallel_gripper_grasp`, `peg_in_hole_circle_loose`.
- A relation id describes possible relations between object ids. Examples are `hi_screw_to_base`, `grasp_point.3_for_hi_screw`, `edge.5_of_base`.
- A relation instance describes an actually established relation between part instances.

**Summary:** The instance level corresponds to the actual workspace situation, the id level of description corresponds to the possible object relations and assemblies, and the type level describes the id level more efficiently.

### 4.3.2 Representation of parts

Every part in the workspace is represented by an instance of the class `partinst`, which is defined as

```plaintext
class partinst
  attributes [  
    inst : atomic_t : [unique,required,unchangeable],  
    id : partid : [required,unchangeable],  
    ordid : atomic_t ]  
identifier inst  
is_a id  
```
The attribute inst uniquely identifies the part among all existing parts. The attribute orderid is instantiated when the part is mated to an assembly. Its value is generated by the scheduler of the M-EAN/S system [Ble92] upon receipt of an assembly order entered by the user. The attribute id establishes a link to the parent class partid which contains a further description of the part. Its definition (abbreviated) is

class partid
attributes [id : atomic_t : [unique,required,unchangeable],
type : parttype : [required,unchangeable],
placetraf : traf_t : [unchangeable],
placeon : objid : [unchangeable],
graspinfo : graspinfo : [multivalued] ]
identifier id
is_a type
rules [on insert
do insertMateidForGrasp(graspinfo,id) ] .

The type specifies an instance of class parttype (not shown here) containing a geometric model of the part. The placetraf and placeon attributes specify if and how the part can be placed on a flat surface or a fixture. The graspinfo attribute specifies one or several instances of class graspinfo, which in turn specifies a gripper, its location relative to the part when grasping, and gripping parameters:

class graspinfo
attributes [id : atomic_t : [unique,required,unchangeable],
gripper : gripid : [req],
matetype : matetype , % type of grasp
grasptpt : point_t , % traf to part
graspclos : numeric_t , % nominal width
graspopen : numeric_t , % width prior to grasp
graspsfrc : numeric_t ] % maximum force
identifier id .

The Prolog procedure insertMateidForGrasp specified in the update rule of class partid will be called when a partid is inserted. It creates, for each graspinfo given, a description of the potential joint between the part and a gripper.

Instances of the classes parttype and partid are inserted at startup time and remain unchanged throughout the operation of the modeling system. Instances of class partinst are created when new objects are observed in the workspace and are deleted when objects leave it for good.

3The attribute name graspinfo and its type graspinfo are the same only by coincidence.
4.3. Workspace model

4.3.3 Representation of relations

Relations between objects model physical contact or observations. We shall call both kinds of relations joints, extending the term used in describing kinematic structures to comprise observations. Potential joints are represented by instances of classes with suffixes “type” and “id”. They are instantiated at startup time. Actually realized joints are modeled by instances of classes ending with “inst”. They are instantiated and deleted during operation of the robot system.

Joints are members of the superclass jointinst which consists of the classes matcinst for physical joints and obsinst for observations. Both classes have essentially the same attributes, but they differ in the hierarchy level at which specific attributes are defined. The need for this differentiation will become clear as the two joint types are described.

Physical joints

Consider two parts mated to one another. Remember that we generalize the term “mating” to mean not only the assembly of parts but any physical contact of interest such as the grasping of a part or its placement on the table or in a fixture. The simplest description of their geometric relation is given by the homogeneous transform relating the frames of the two objects, along with its uncertainty. A further description of the relation may include the type of connection or assembly operation, tolerances, mating trajectories and the mechanical stability of the completed joint. In order to allow for generic mating strategies, it is especially desirable to classify joints into a limited set of joint types.

A possible classification of assembly operations, together with their relative frequencies, is given in [NW80]. The dominating operation is the peg in hole operation with its variants (inverse, multiple and generalized form peg in hole). It has been analyzed intensively [Whi82]. It comprises a great variety of tasks, ranging from the insertion of a shaft into a bearing to the placement of a cover onto a finished assembly. It is also used by other operations, e.g. in the first stages of a screwing operation, a force fit or a snap joint.

In ARGUS, peg in hole joints can be further classified according to the freedom of motion left between the mated parts. This allows to model the effects of mating operations on positions and uncertainties, as described in section 4.1.

The description of the geometric relation of two parts by a single UT is not sufficient: Uncertainties and tolerances are best described and reasoned with in a frame associated with the joint itself, and not in the frames of the joined objects. We shall therefore relate object frames via the joints connecting the objects. The total transform is divided into the transform from the top object to the joint, and the transform from the joint to the bottom object. They describe the nominal relation between the objects. Due to uncertainties it will differ from the actual relation. The difference is expressed by an additional transform from a joint frame attached to the top object to one attached to the bottom object (Fig. 4.12). The uncertainties used in the preconditions
of operations and when modeling the effects of operations can now be expressed and interpreted naturally in the joint frames.

**Physical joint attributes**

We use the following conventions: Two related objects are called the *top* and *bot* (tom) object of that relation. The joint frames associated with the two objects are labeled *topjoint* and *botjoint*. The arrows of the transforms relating them always point from the top to the bottom frame. The transforms are called \( \text{top2joint} \), \( \text{jointtraf} \) and \( \text{joint2bot} \), and each has an associated uncertainty, \( \text{top2joint.unc} \), \( \text{joint.unc} \) and \( \text{joint2bot.unc} \). Of these, \( \text{top2joint.unc} \) and \( \text{joint2bot.unc} \) express manufacturing uncertainties in the location of joint surfaces of each object. They are given in type or id definitions and are unchangeable. The \( \text{joint.unc} \) exists only for actual relations and may be updated when modeling.

Since the description of uncertainty depends on the frame of reference, it has to be specified for these uncertainties. The following convention is obeyed:

<table>
<thead>
<tr>
<th>uncertainty</th>
<th>frame of reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{top2joint.unc} )</td>
<td>( \text{topjoint} )</td>
</tr>
<tr>
<td>( \text{joint.unc} )</td>
<td>( \text{botjoint} )</td>
</tr>
<tr>
<td>( \text{joint2bot.unc} )</td>
<td>( \text{botjoint} )</td>
</tr>
</tbody>
</table>

One might as well describe \( \text{joint2bot.unc} \) in the \( \text{bot} \) frame, but defining uncertainties in the same frame shortens the computation of the total uncertainty of the joint. The same holds for \( \text{top2joint.unc} \).

In addition to transforms and uncertainties, a physical joint is described by the following attributes:

---

\(^4\)Normally, but not necessarily, the top object is "on top" of the bottom one in the everyday sense
4.3. Workspace model

**strength**: The behavior of the joint as forces are applied to it. Currently, it may take the values "fixed", "loose" or "fragile".

**uncbefore**: The maximum uncertainty allowable for successful mating by straight line insertion, subject to the constraints of section 4.1.1.

**max_joint_unc**: The uncertainty after successful mating, imposed by the joint itself. It is used to derive the joint constraint matrix and to update the joint uncertainty after mating.

**topid, botid**: The object ids being related.

**approach, getaway**: A sequence of straight line motions for mating and unmating.

**graspsinfo**: When inserting a partid, mateids representing potential grasp points are inserted, using graspsinfo data. The value of this attribute points back to the graspsinfo from which this partid was created. When a particular mateid is selected for grasping, the corresponding grasp parameters can be accessed via this pointer.

**temp**: Normally, joint ids, describing potential joints, are loaded at startup time and remain unchanged. But it is possible to place or detect an object at an unforeseen position in the workspace, for which no mate id exists. In such cases, a temporary mate id must be created before the mate inst representing the real joint can be inserted. When later deleting the inst, its id will be deleted automatically if temp is "true".

**topinst, botinst**: The object insts being related.

**abs_unc, ref_frame, ref2botjoint, changed**: During the planning process, the total joint uncertainty, expressed in some reference frame, will be computed repeatedly. If parts of the relation network remain unchanged, then the computational effort can be reduced. These attributes save relevant data and indicate whether the joint has been changed since the last computation.

These attributes are defined at different levels of the class hierarchy. Some are not required at the inst or id level and therefore have defaults given at the type level. The levels at which attributes are declared, together with the attribute characteristics, are given in table 4.1.

**Observations**

An observation relates a sensor to an observed feature. In general, it also may create relations among features, but we shall not model such relations explicitly. If necessary, they can be represented by two observations relating the features via the sensor. Similar to physical joints, observations are represented by triplets of transforms relating sensors to objects. The three transforms are: From sensor to observed feature, from observed to actual feature and from actual feature to object frame (Fig. 4.13). The difference between observed and actual feature was introduced (see section 4.1.7) since the pose of a feature usually provides only partial information about its object's pose. The information on the transform from the actual feature to the object frame is
complete. The information from the observed feature to the actual feature is partial, depending on the type of the feature. The information from the sensor to the observed feature is partial, depending on the type of the feature and the sensor.

Observation attributes

Most attributes carry the same meaning as with physical joints. See table 4.1 for levels and characteristics. Differences lie in the origin of joint attribute data and in the class hierarchy level at which it is represented. The important attributes are:

*top2joint* is the transform from sensor to observed feature, corresponding to the observation data. It is known only after an observation is made, and therefore is defined at the instance level. If the observation does not completely determine the transform, the uncertainty *top2joint.unc* will reflect this.

*jointtraf* and its uncertainty *joint.unc* are properties of the feature. They relate the observed to the actual feature. The transform always is the identity transform. The uncertainty expresses the constraints that the pose of the observed feature imposes on the pose of the actual feature. This corresponds to the motions under which the appearance of the feature does not change: A straight line can be rotated about and translated along itself, or a plane can be rotated about its normal and still look the same. The two attributes are defined at the type level.

*joint2bot* and its uncertainty *joint2bot.unc* state the relation from feature to object. They depend on the object model and are represented at the id level. The uncertainty usually is zero or small.

*strength*: It always has the value "fragile" and therefore is defined at the type level.

Figure 4.13: Observation representation. The discrepancy between the observed and the actual feature pose is modeled by a transform which nominally is the identity transform and whose uncertainty reflects the constraints between the two poses.
featdata contains feature specific data. For example, the type of a feature will specify that it is a line, and the featdata attribute will contain the length of the line.

4.4 PLANNING

The issues involved in planning were introduced in section 2.4. Here we show a description of robot system capabilities that allows to plan actuator and sensor operations. The description is an extension of the traditional operator description [HTD90]. It is meant to be of a general form, such that it can be used by different planners. Since the main goal of this work is to demonstrate the use of uncertainty modeling for planning, a relatively simple hierarchical planner is implemented.

4.4.1 The agent concept

As seen in section 2.4, an important question when designing a planning system is if and how to modularize the planning process. We choose to distribute planning knowledge over a set of agents that achieve specific operations, and an operations planner which reasons about the agent’s capabilities. An agent is an entity consisting of hardware and software that is able to execute a specific operation. This is an extension of the software concept of program modules. An agent contains operation specific algorithms for local planning, processing sensor data and controlling actuators. It may rely on other agents and use an external database to provide information. Several agents may use the same hardware resources. Agents implement object level operations such as grasping, moving, mating and observing objects. Since the set of operations used in assembly is limited and known, it is possible to have an operations planner which reasons in terms of these operations, but is independent of the implementation of the operations. The agents and the operations planner use a common database containing object-specific geometric and physical information and the current workspace state. This approach promises the following advantages:

- Local planning by agents is coupled only loosely to operations planning, simplifying the planning process.
- The operations planner can be modified independently of the agents.
- Agents can be modified or added without changing the operations planner, and agents can be developed independently from one another.
- The operation planner’s selection of agents can be based on agent descriptions alone, without having to know the details of agent operations.

The approach also leads to the following problems and requirements:

- The mutual dependencies of agents must be expressed in a standard way and processed by the operations planner.
• Should the applicability of an agent in a given situation be evaluated by the agent or by the operations planner? On the one hand, the required knowledge is agent specific and should be incorporated in the agent. On the other hand, the operations planner may have to compare the applicability of different agents.

• How do agents and the operations planner interact with the workspace model: Who retrieves data and who updates the workspace model?

• An agent must be able to decide whether it has succeeded in executing its operation or not, and the operations planner must be able to cope with the failure of an agent.

4.4.2 Agent description

In the classic operator based planning systems, planning is based on logical relations between objects, which can be represented by predicates that are either true or false. The changes that an operation makes to a blocks world model are represented by so-called add- and delete-lists containing the predicates that are added to or deleted from the world model. A grasping operator would be described by

\[
\begin{align*}
\text{operator} & : \text{grasp(A)} \\
\text{precondition} & : \text{free\_top(A), gripper\_empty} \\
\text{add-list} & : \text{grasping(A)} \\
\text{delete-list} & : \text{supported(A), free\_top(A), gripper\_empty}
\end{align*}
\]

Using this information, a planner can use forward or backward reasoning to find a sequence of operators that achieves a given goal. In a system applied to real assembly tasks, the workspace state is not sufficiently represented by logical predicates alone, and planning needs to consider the geometric relations between objects.

We shall include uncertainty preconditions in the description of agents and implement the logical aspects of planning in a hierarchical planner. The latter is reasonable, since

• we are mainly interested in the role of uncertainties in planning,

• the M-EA$$\text{NS}$$ system generating commands for ARGUS takes care of the planning and scheduling of object level operations to achieve the global assembly goal,

• the purely logical aspects of planning a single object level operation are simple.

An agent is described by its

\text{type}, denoting the type of operation implemented by the agent, including parameters.

\text{id}, which differentiates between several agents of the same type.

\text{logical precondition}, containing predicates that can be evaluated by the operations planner. They express conditions on the workspace state which must be satisfied in order for the agent to be applicable. The logical precondition represents \text{declarative} knowledge which could be used to reason backwards when planning.
uncertainty precondition, a procedure that checks uncertainties relevant to the agent's applicability. It contains procedural knowledge inaccessible to the planner. It returns a succeed/fail flag and agent specific data that can be used to fix the plan such that the precondition is satisfied.

expansion, expressing how the agent is decomposed into procedures for information retrieval, acting, sensing and updating the workspace model, as well as into other agents. This last capability allows to define a hierarchy of agents.

The description of an agent must give its type and at least one of the other four items. As an example, two agents used by ARGUS are shown. Some of their aspects will be explained later:

```plaintext
is_agent(
  type : grasp(Gr,A),
  id : direct_grasp(Gr,A),
  lprec: not(holding(Gr,X)), % gripper is free
  expand: ( findJointId(Gr,A,Jid,_),
    move_to_grasp(Gr,A,Jid), % do action
    calcUncForApproach(Gr,A,UncGr,UncA), % compute info
    approach(Gr,A,Jid,UncGr,UncA), % call other agent
    activateGripper(Jid), % do action
    create_phys_joint(Gr,A,Jid) ) % update model
).

is_agent(
  type : approach(T,B,Jid,UncT,UncB),
  id : blind_approach(T,B,Jid,UncT,UncB),
  lprec: blind_app_uprecd(T,B,UncT,UncB,FixInfo,Succ),
  expand: ( getApproachDMove(Jid,DM), % compute info
    cg_dmove(T,DM) ) % do action
).
```

The main acting agents implemented in ARGUS are the (un)grasp and (un)mate agents. They are agents in the strict sense, as described in section 4.4.1. Thanks to the expressive power of the agent description, planning knowledge on the operations level and inside agents can also be expressed in terms of agents. Fig. 4.14 shows the relation of agents as implemented in ARGUS: Planning agents decompose object level commands into calls to agents in the strict sense. These in turn are implemented in terms of primitive agents. The planning and primitive agents could be replaced by dedicated procedures, but the uniform description allows for greater clarity and flexibility.

With the structure of agents as in Fig. 4.14, uncertainty preconditions of acting agents can be restricted to a single agent, i.e. the approach agent. The precondition for a specific mating strategy depends on
Figure 4.14: Acting agents and their calling hierarchy. An arrow $A \rightarrow B$ indicates that agent $A$ may use agent $B$. The clear_top, free_gripper and free_joint agents take care of unwanted parts in the gripper or on the parts to be handled.
4.4. Planning

1. the a priori relative uncertainty of the parts to be mated
2. the type of joint to be established
3. the maximum uncertainty that can be passively absorbed by the joint
4. the mating strategy itself

Item 1) can be extracted from the current workspace state, items 2) and 3) are constant properties of the joint, and item 4) is determined by the agent and may be influenced by the other three. How should this data be processed? We wish to satisfy the following requirements:

- The implementation of the approach agent should be independent of the joint parameters, but not necessarily of the joint type. For example, one agent might be applicable to loose circular peg in hole joints of different sizes, another one to peg in hole joints requiring a force fit.
- The planner must compare the uncertainty preconditions of different agents of the same type. It does not suffice to have the planner check the relative uncertainty of the objects to mated, since then the mating strategy is not accounted for. This can only be done by the agent itself, which therefore must check its own preconditions.
- If an agent's uncertainty preconditions are not satisfied, it should return information to guide a plan fixer which tries to change the plan to make the agent applicable. Therefore not the relative uncertainty of the parts being mated is determined, but the uncertainty of each part with respect to the world frame. This allows a fixer to determine which uncertainty (if not both) has to be reduced.
- Agent specific knowledge should be restricted to the agent. This requirement cannot be satisfied completely: First, the computation of prior uncertainties is separated from the approach agent, such that different approach agents will not have to repeat the same computations. Second, the uncertainty precondition checker of the approach agent returns data which must be processed by a fixer that is separate from but specific to the approach agent.

The result is that the approach agent takes the joint id and the a priori uncertainties of its objects as parameters, and that the uncertainty computations and the fixing strategies are common to all approach agents. The uncertainty computations are done by the agent calling the approach agent, i.e. by the grasp and mate agents.

4.4.3 Planning the use of acting agents

The planner accepts an object level goal and generates a sequence of calls to acting and sensing agents which achieve the goal. Since the domain specific planning knowledge is expressed by the planner agents, the planner can be relatively simple: Possible goals correspond to agents which expand to other goals. Given a goal, the planner searches for the first agent of matching type and checks its logical precondition. If the check
fails, the search continues. If no fitting agent can be found, the planning process fails. If the check succeeds, the uncertainty precondition (if it exists) is checked. If it fails, an agent specific fixer could use the current plan and failure data to create a fixed plan such that the uncertainty preconditions are satisfied.

It still is necessary to choose among different agents of the same type but with different uncertainty preconditions. In the current implementation, uncertainty preconditions are not evaluated. They could be compared by the following strategy: Order agents of the same type such that the one with the least restrictive preconditions is tested first. This agent will be the most complicated one. Since it may be more sophisticated than required, test the next agent until one is found that does not satisfy the uncertainty preconditions without having to fix the plan. Select the agent tested just before this one. This heuristic procedure ensures that uncertainty reduction is done locally (i.e. by the agent itself and not by adding fixes earlier in the plan) whenever possible, and that the simplest agent possible is chosen.

If an applicable agent has been found and its expansion exists, the expansion is processed. It may contain Prolog procedures or further agent calls. Procedures for retrieving simulator model data or updating the simulator model are executed, and retrieved information is used in subsequent procedure or agent calls. Agent calls are added to a list of goals to be satisfied. For an example, see section 6.3.

**Workspace model maintenance**

During planning, it is necessary to keep track of the goals to be satisfied, the plan created so far, and the workspace state represented by the planner model. When planning begins, the model state represents the real workspace. This state shall be called to *initial state*. During planning, the model is continuously updated by the simulation of primitive agents. The model is queried by the logical and uncertainty preconditions and provides position information for motion commands. After a plan has been created, it can be executed, using the real robot (or a simulator model). Before the execution, the workspace model must correspond to the initial state. It then is updated by the execution of each plan step.

We use the same modeling procedures for both planning and executing plans, and save and load the workspace state before and after planning. This is done by writing the Prolog clauses representing the workspace state to a text file that can be reconsulted. In this way, a workspace state can also be saved for a separate session with the ARGUS system.

When fixing a plan, planning will continue from a state in the middle of the current plan. How can this state be represented efficiently? It clearly is not sensible to save the entire workspace state after each step of the plan, since only a small part of the model is changed at each step. Trading memory against execution time, one may simply load the initial state and re-simulate all the actions up to the state of interest. We choose an intermediate approach: All update operations to the model are recorded. Since the model is implemented in the knowledge model, this is done by the three knowledge model update operations. When a given state must be restored, an earlier
state is loaded, and the update operations up to the given state are repeated. This straightforward execution of a list of update operations saves the time to duplicate the replanning effort.

4.4.4 Plan representation

A plan generated by applying the grasp and approach agents which were shown earlier earlier is represented by the following commented hierarchy of agents and procedures (for readability, most arguments are not shown). Each agent is followed by an indented sequence of its subgoals and/or Prolog procedures:

```prolog
grasp(large cyl_1) % the planning goal
  grasp(sgrpper, large cyl_1) % selected gripper
  findJointId % get new joint info
    move_to_grasp % move to approach point
      getApproachPt % compute approach point
      getGripSettingBeforeGrasp % get gripping data
      gross_move_grpos % execute the move
    calcUncForApproach % compute relative uncertainty
      approach % move gripper to part
        getApproachDMove % get approach move data
        cg_dmove % execute the move
      activateGripper % close gripper
    create_phys_joint % update workspace model
```

For the execution of the plan, the sequence of procedures is sufficient. In the above case it is

```prolog
findJointId
getApproachPt
getGripSettingBeforeGrasp
gross_move_grpos
calcUncForApproach
getApproachDMove
cg_dmove
activateGripper
create_phys_joint
```

The goal hierarchy is maintained anyway, such that a fixer has more information about the intentions of the plan than can be derived from the procedures alone. For example, if a fixer wants to insert an observation of a part just before it is grasped, it must determine that a certain sequence of procedures implements a grasp operation. This amounts to reconstructing the meaning of the plan, as in [SG86], and is greatly simplified if the goal hierarchy is available.
4.4.5 Choosing sensors and features for sensing

A fixer first determines at which point in the assembly process some specific pose information about an object must be acquired. Then it must select sensors and features whose observation will provide the needed information. This may be done by heuristic procedures, based on a numerical analysis of the involved uncertainties, and may be verified by simulation. For example, the following statements informally express heuristic knowledge (for the two-dimensional case) about the relation between sensors, features and objects:

- The information a camera provides about a linear feature is proportional to the sine of the angle between the viewing direction and the feature.
- The information that a linear feature provides about the translation of an object in a given direction is proportional to the sine of the angle between the feature and that direction.

In ARGUS, the feasibility of such an approach is demonstrated by the implementation of a fixed heuristic procedure, limited to the two-dimensional case: It is given the name of an object and a maximum linear uncertainty to be tolerated, expressed in the object's frame. It selects edge points on a two-dimensional object contour, whose observation reduces the uncertainty in the desired way. It works as follows (See section 6.5 for an example):

1. The current linear uncertainty of the object is determined and represented in the object's frame. This uncertainty is compared with the maximum uncertainty (See section 3.2). If it is too large, the needed information is computed. Otherwise, no observation is required.

2. The upper left submatrix corresponding to translations is extracted from the needed information. The direction in which most information is required is given by its eigenvector with the largest eigenvalue. Let a vector orthogonal to this eigenvector be $x_{\text{max}}$.

3. The edges of the object contour are collected and their angles to $x_{\text{max}}$ are computed.

4. The edges are sorted by ascending absolute angles. Edges with equal angles are sorted by descending lengths.

5. The first edge in the sorted list is selected for observation. If its absolute angle does not exceed a certain threshold, it provides enough information, and the feature selection is complete.

6. Otherwise, the next edge in the list is selected whose angle has the opposite sign than the angle of the first edge. The combination of these two edges gives the maximum information about the required direction that can be obtained from two observations.
Another strategy for determining an object's pose by observing points on its contour has been implemented independently of ARGUS. For an object whose pose on a plane surface is known to within a few millimeters, it determines a set of three points whose observation gives a better estimate of its 2D location and orientation. It is limited to objects that can be described by the union of arbitrarily sized cubical blocks with parallel sides (Fig. 4.15). The main part of the procedure involves the computational geometry problem of determining the object's 2½D contour from its description. As in the preceding algorithm, contour features are of interest because they are easy to detect with optical sensors and a suitable background. The 2½D contour contains the highest outermost edges and can be detected by a distance sensor looking downward. Once the 2½D contour is known, two points that lie on parallel edges and are separated by the largest distance are determined. They provide the maximum information (obtainable from two points) about the object's rotation, as well as translation information in one direction. A third point on a perpendicular edge provides the information on the other linear axis. All observed points are checked whether they allow a collision free operation of the sensor, i.e. a certain volume around the observed points may not be occupied by the object.

As a side effect, the object contour analysis also finds grasp points for a parallel gripper and determines the height of the object.
Table 4.1: Attributes of physical joints and observations. An entry denotes the existence of an attribute at a specific level of description. The letters q, r, u, and n indicate that an attribute has the unique, required, unchangeable or no special characteristic.
One of our goals was to implement the ARGUS/M-EANS system in a real environment. This entailed a large amount of work used to create a working assembly system, starting from scratch. A number of system functions were developed and implemented in depth, as described in this thesis and in [Ble92]. Other functions vital to a working system were realized in a more primitive fashion. They include solid body modeling, gross motion planning, force-controlled parts mating, error detection, keeping track of which parts of the table are occupied, and the calibration of the manipulator, the gripper and the sensors.

In this chapter we first present the overall structure and the interfaces (including user interfaces) of the functions that were introduced in the preceding chapters. They are specific to ARGUS. Then the implementation environment is described, including low-level software that is specific to individual agents, and Prolog tools whose use is not restricted to ARGUS. At the end, the limitations of the current implementation are listed.

5.1 SYSTEM DESIGN

The building blocks of ARGUS are shown in Fig. 5.1 and will be described in this section.

5.1.1 Network functions and operations modeling

The network functions realize the computation of relative uncertainties and the integration of observations into the network of relations. The two procedures available for outside use are called relUnc and updateNetDesc, as described at the end of chapter 3. The procedure relUnc retrieves logical and geometric data from the workspace model to assemble the recursive description of the relation between two nodes. The procedure updateNetDesc takes this description apart again and makes changes to the workspace model.

The procedures create_phys_joint, delete_phys_joint, change_phys_joint and create_obs_joint model the effects of mating, unmating, moving and observing oper-
5.1.2 Workspace model

The workspace model describes the possible and actual parts and relations. All information is accessed through the knowledge model's update and retrieve operations, with one exception: A lot of graph searching is done when computing uncertainties and when determining the logical or geometric relation between parts. This requires network topology information to be retrieved very often, making the performance of the knowledge model's retrieve operator crucial to the overall execution speed. Therefore, the purely topological information contained in instantiated joints is asserted by additional Prolog clauses, which are used by the graph searching algorithms. They have the structure

\[
\text{joinedInst} \left( \text{JointInst}, \text{TopInst}, \text{BotInst}, \text{Strength} \right)
\]

where the arguments have the same value as in the corresponding jointinst (see table 4.1). The clauses are automatically asserted and retracted by the consistency rules of joint instance classes.

The workspace model as a whole is managed by procedures for, on the one hand, reading object and workspace description files and, on the other hand, for saving and reconstructing selected model states by the planner. The model description files are created manually and contain knowledge model insert commands. For examples, see chapter 6.
5.1. System design

When the world changes, the question is who should update the model. Should changes be made e.g. by a grasp operation or, after its successful execution, by the planner that called it? We adhere to the rule that updates should be done locally by the agent or the robot level procedure that causes a change. This has the disadvantage that all these procedures are model dependent, but the advantage that procedure specific knowledge is localized in the procedure itself. The dependency on the model does not weigh so heavily, since the model, being vital to the entire system, should be comparatively stable.

As a result, the different system functions update the workspace model in the following way: The procedures in the robot/simulation interface implement elementary moves. They therefore also update the model to reflect the robot and gripper positions after the move. The acting agents implement (un)mating and moving operations. They therefore update uncertainties and transforms changed by the addition or removal of physical relations. The observation agents collect information and insert it into the model, using the update procedure provided by the network functions.

The state of the workspace can be displayed textually or graphically:

Textual display

On the knowledge model level, the operators dispcl, dispid for displaying class and instance information have already been introduced. For example, a list of all instantiated jointtypes is obtained by typing

?- dispcl jointid. % entered by the user

Superclass jointid contains the classes [mateid, obsid]

Class instances of class mateid
  dispid 'tempTableMateId3_for_demo_object'.
  dispid 'tempTableMateId1_for_obstestpart'.
  dispid 'endofarm2irsensor'.
  dispid 'world2camera'.
  dispid 'endofarm2sgripper'.
  dispid 'world2endofarm'.
  % ... etc

Class instances of class obsid
  dispid 'edge6'.
  dispid 'edge5'.
  % ... etc

In the output text, each name is preceded by the operator dispid. This is not informative, but useful: In the Macintosh Programmers Workshop (MPW), under which ARGUS is developed, pressing the ENTER key sends the line containing the cursor to
the Prolog interpreter. Therefore, clicking on one of the ids in the output and pressing ENTER will display its attributes and their values.

The structure of relations is displayed with the command disptree: The network of joint instances is represented as a tree with the world frame at the root. Each object is followed by an indented list of the objects to which a relation exists. The strength (fixed, fragile, loose) of the relation is indicated by a full, dashed or dotted line, and the description vector of the total transform over the relation is given. Loops are indicated by marking relations to a node that already is in the tree with a '+' sign. At that node, the relation's inverse will be displayed. For example (the output is truncated manually):

?- disptree.

```
_ _ largeblock_1 ............[-400.0,200.0,-400.0,-0...
+----- camera ............[377.5,435.0,1000.0,0,0,0...
- smallcyl_1 ............[140.0,35.0,10.0,-0...
+----- camera ............[160.0,365.0,990...
_ _ smallblock_1 ............[-400.0,300.0,-400.0,-0...
- camera ............[-200.0,400.0,600.0,0,0,0,0,0...
+----- smallcyl_1 ............[-160.0,-365.0,-990...
+----- largeblock_1 ............[-377.5,-435.0,-1...
- endofarm ............[100.0,100.0,102.0,0,0,0,0,0...
- irsensor ............[0.1,25.0,-309.4,0,0,0...
- sgripper ............[0.1,4.6,-309.4,0,0,0,0,0...
- SGRrightFinger ............[-7.5,3...
- SGRleftFinger ............[-7.5,-23...
- SGRBody ............[-70.0,-48.5,87...
```

Here, parts largeblock_1 and smallblock_1 are lying loosely on the table. The part smallcyl_1 is affixed rigidly to largeblock_1. The camera, endofarm and peripherals are affixed to the world frame. There exist loops via the two observed parts smallcyl_1 and largeblock_1. The relation from e.g. smallcyl_1 to camera is shown twice, once with respect to smallcyl_1 and once with respect to camera.

As in the preceding example, the output lines are executable Prolog commands, which here would display the information on the joint instances.

Graphic display

Some graphic visualization of the workspace state clearly is required. On the one hand, one wants to verify the manually input models and the robot positions generated when simulating, in order to prevent unpleasant surprises when working with the real robot. On the other hand, one wants to visualize the uncertainties resulting from the different operations modeling procedures.

While the M-EANTS system uses a solid body modeling system to detect the intersection of objects, the current version of ARGUS requires object models only for display
purposes. The extraction of geometric features used for sensing is done manually. As a first step, it therefore is sufficient to represent an object by a set of wireframe models of primitive objects (cubes and cylinders). Their Prolog representation is a list of primitives and their pose with respect to the object’s frame. This description is stored in the workspace model under the model attribute of the class objtype. A primitive may also be used to display a cylindrical or cubical em hole, as in the following example, where an object type with the model shown in Fig. 5.2 is created.

\begin{verbatim}
insert parttype with [
    type : demo_object,
    model : [ 'cube([70, 50, 30]),
              'cube([120, 20, 20], [70, 25, 5]),
              'cyl([10, 15], [35, 25, 15]),
              'cyl([5, 20], [165, 45, 15, -pi/2, 0, 0]),
              'cyl([5, 20], [165, 5, 15, -pi/2, 0, 0])
            ].
\end{verbatim}

In the cube(Desc, Traf) and cyl(Desc, Traf) structures, Desc is a parameter vector defining the size of the cube (x/y/z extensions) or cylinder (radius/height) in millimeters, and Traf is a description vector or a transform giving its relation to the part frame. Its entries may contain the symbolic variable pi, trailing zero entries in the description vectors may be omitted, and the identity transform may be omitted altogether.

For the display of models, the graphic facilities of Matlab are used. Information is passed from ARGUS to Matlab by two files: One file containing object type descriptions is processed once after starting Matlab. A second file containing the current state of the workspace, i.e. the identity and location of instantiated objects, is generated by ARGUS and displayed by Matlab, using a set of Matlab functions written by
the author. Viewing angles, distance and magnification can be modified interactively within Matlab.

The display system works basically as follows: From the descriptions of primitives, Matlab creates a matrix containing a sequence of homogeneous coordinate vectors of the primitive's vertices. This matrix is multiplied with the transform giving the position of an object, and with the viewing transform (containing the camera position and focal length [Pau81]). The result gives the object vertex coordinates in the image plane. The straight lines connecting these vertices are drawn with the Matlab plot command.

For visualizing uncertainties, separate 3D crosshairs are drawn for linear and angular uncertainties. The uncertainties are diagonalized by the procedures developed in section 3.1.5. The lengths of the crosshair lines are either proportional to the uncertainty ellipsoid's main axes, or may be scaled logarithmically.

5.1.3 Planner and object level interface

The planner itself has no connection with the remaining system, except that it must be able to save or load the workspace model state. The other connections are established via the agent descriptions (see section 4.4.2). The descriptions are processed at startup time and converted to an internal representation. They contain the names of procedures for retrieving information, modeling operations and applying sensors and actuators. The procedures are called when planning or executing plans.

ARGUS provides a set of object level commands and, based on them, a set of functions that are used by M-EANS. The object level commands correspond to the available agents. The main command is

\[ \text{'puton'} ( \text{Top\_Object} , \text{Bottom\_Object} ) \]

where \text{Bottom\_Object} may also identify a fixture or a position on the table, or may have the value anywhere. The planner decomposes the 'puton' command into 'move', 'grasp', 'mate', 'ungrasp' and similar commands, which may also be entered individually.

The scenario, when operating ARGUS in conjunction with M-EANS, has parts enter the workspace in an area observed by a camera. They are moved to a buffer area or, if possible, assembled immediately in an assembly area. The online scheduling implemented in M-EANS uses the following interface procedures provided by ARGUS. They manage the instantiation and deletion of parts and call the planner. Each procedure has an output argument \text{ReturnStatus} which may signal errors such as missing parts, the failure of an operation, or the malfunctioning of a hardware device.

- \text{means\_newtypes} ( -\text{TypeList}, -\text{ReturnStatus} )
  activates the camera and returns a list of part types that were not observed previously. This implies checking, for each observed object, whether an object of the same type is already registered in the workspace model at approximately
the same position. If there is one, its position is updated; if there is none, it is instantiated and a temporary physical joint id and instantiation relating it to the world frame are created.

- **means.buffer( +TypeList, -ReturnStatus )**
  moves the parts whose types are given in TypeList from the vision area to the buffer area. Parts that do not require a dedicated fixture are allocated an arbitrary free location on the table.

- **means.mate( +Partid, +Orderid, -ReturnStatus )**
  assembles a part with the given id and assigns it the order id, which is unique for each assembly or subassembly.

First, since only the part id is given, the part instance of the top part must be selected: Recall that parts of the same type may be used at different locations (identified by part ids) of the same assembly. Therefore, a part instance with the same type as the given part id is sought. It may not have been assigned to another assembly, and parts in the vision area are used prior to parts in the buffer area.

Second, the instance of the bottom object of the mating operation must be chosen: If there is part instance with the same order id, to which the top part can be mated, then it is selected. If no such part exists, then the top part is assumed to be the first of an assembly and is placed in the assembly area.

- **means.put_in_output_buffer(+Objid,+Orderid,-Status)**
  moves an assembly to an output location and deletes all its part instances and the instances of any joints connected to them.

### 5.1.4 Robot interface and simulation

The interface to robot commands is divided into three levels (See Fig. 5.3). On the first level, motions are specified in terms of objects. The following procedures are available:

- **gross.move( +ObjectInst, +Transform )**
  moves the frame of the given object instance to the specified absolute frame, assuming that the object is affixed directly or indirectly to some gripper. The object may, of course, be the gripper itself. The object’s pose relative to the manipulator end of arm is determined, which allows to compute the required end of arm position. That position is passed to a second level procedure.

  A similar command **gross.move.grpos** takes an additional argument which gives the desired gripper position at the end of the move. It saves execution time when moving to grasp a part.

- **cg.dmove( +ObjectInst, +Transform )**
  executes a compliant and guarded move relative to the current object position.\(^1\)

\(^1\)A guard is a condition on positions or forces which, when satisfied, terminates the move.
The given object instance might be used to retrieve joint specific trajectory and compliance parameters, but currently a single mating strategy is applied to all joints (see section 5.2.1). A related procedure gross_dmove does a relative move with collision detection only.

- `activateGripper(+JointId)` retrieves grasp data associated with the given joint id, which may include the id of the part to be grasped, the gripper being used and grasp parameters. It calls a second level procedure with gripper specific data, currently the gripping force and the expected width after grasping. The latter will be compared with the actual width. The gripper is opened again with `deActivateGripper`.

On the second level, commands are at the device level: Commands defined at the third level are executed, and their return information is used to update the device's representation in the workspace model.

On the third level, commands are executed either by activating the real robot or by simulating their outcome. The final, real or simulated position of the manipulator or gripper is returned. A software switch allows to select the mode.

Calls to the actual robot are executed by assembling text strings from command names and parameters, which are sent to the robot controller (see next section) by serial interface. After execution, the return string is parsed, giving either position data or the name of a controller exception. If an exception is detected, an associated Prolog exception is raised. When working with M-EA\(_{\text{NS}}\), the exceptions will be handled in the M-EA\(_{\text{NS}}\) interface, if not earlier.
5.2. Implementation environment

5.1.5 Utility functions

In addition to the knowledge model and matrix functions (see next section), there exists a number of utility functions that are used everywhere in the system: One group of functions allows to create and delete part instances conveniently and to keep track of occupied table positions. Another group implements graph searching algorithms for finding a path between two given nodes, for finding the lowest part in a collection of grasped parts, for determining the sets of affixed or mated nodes, together with the joints leaving the node sets (defined in section 4.1.4), and so forth. A third group provides elementary procedures for manipulating transforms and uncertainties, and for computing the total transform and relation of a joint from its three constituent UTs.

5.2 IMPLEMENTATION ENVIRONMENT

The functions presented in the above sections all are implemented in the Prolog language, running on a Macintosh PC. The latter is connected by a serial interface to an industrial IBM PC AT computer, around which the various hardware devices are clustered. Software running on the IBM PC is written in the programming language AML/2. It controls the manipulator, communicates with the other devices and implements primitive agents.

5.2.1 Hardware and low level software

The hardware basis of ARGUS is an assembly system consisting of an IBM 7575 SCARA (Selective Compliance Assembly Robot Arm) manipulator with four degrees of freedom, an industrial IBM PC AT computer with dedicated control hardware and a servo power module (SPM). It is equipped with a servo controlled gripper, a force/torque sensor, infrared reflectance sensors and a vision system (Fig. 5.4). The devices are linked by RS-232 serial interfaces.

Force/torque sensor

A six-axis force/torque sensor (FTS) from Assurance Technologies Inc. (formerly LORD Industrial Automation) [Aut86] is mounted in the robot wrist. It has a dedicated controller that returns force/torque data over the serial interface. The range/accuracy/resolution of measurements is ±66N/0.53N/0.05N for forces and ±5.6Nm/45Nmm/1.4Nmm for torques. It is possible to specify the coordinate frame in which the forces and torques should be represented (usually the grasp point is chosen). Several so-called "force/torque conditions" may be stated and associated with signal lines on a parallel port. Satisfaction of a condition will then activate the corresponding signal. The sensor is used for several purposes:

Safety: A basic safety mechanism is realized at the hardware level: The overload signal line provided by the FTS controller is connected to the emergency stop
Figure 5.4: Hardware structure. VS: Vision System, SPM: Servo Power Module, FTS: Force/Torque Sensor, SGR: Servo Gripper, AD/DA DIDO: Analog and Digital Input/Output. (Figure: courtesy of R. Bless.)
5.2. Implementation environment

circuit of the manipulator. If the sensor is overloaded, malfunctioning or without power, manipulator motion is stopped immediately and its power supply is shut off.

Collision detection: If initialized properly, a set of force/torque conditions is declared that detects values in excess of 70% of their maximum. The corresponding signal line is connected to the robot controller where it is continually monitored by an AML/2 monitor process. Upon activation, the monitor stops the manipulator (without powering it down) and raises an exception. If not handled within AML/2, the exception is passed on to the calling Prolog procedure.

Compliant motion: The forces and torques corresponding to the manipulator's degrees of freedom are used to control its position. Motion is specified by a gain vector \( g \), a bias vector \( b \), and terminating conditions on forces and positions [Str87]. In the control loop, the torque/force vector \( f \) is measured, and the change in position \( d \) is determined as

\[
d = (f + b) \circ g
\]

where \( \circ \) is the element-wise product. Sequences of compliant motion commands have been used to implement peg in hole mating operations, including the search for mating positions. Because of the slowness of the AML/2 interpreter, the sampling rate of the control loop is around 5-10 Hz.

Heuristic mating strategies: We implemented a classic strategy for mating parts whose misalignment only slightly exceeds the error absorbed by chamfering. The mate operation is accomplished by moving the top part in the negative \( z \) direction. Upward forces arising due to contact are monitored. If they exceed a certain threshold, the move is stopped. If the part now is near its goal height, the mate is assumed to have succeeded. If not, a small amplitude "wobble" motion in the \( x/y \) plane is performed while applying a small force in the negative \( z \) direction. Then a further downward move is attempted. The procedure is repeated until the goal position is reached, or until a given number of trials is exceeded. In the case of failure, an exception is raised.

Gripper

A two-fingered parallel servo gripper is controlled by specifying either the gripper force or its position. Grasping is done with a force command. After its completion, the gripper position is queried. If no part was grasped, or if a grasped part is lost, the gripper controller detects and signals an error.

Infrared reflectance sensors

Each gripper finger incorporates a fiber optic reflectance sensor looking downward. It emits a modulated beam of infrared light and measures the intensity of the reflection. The intensity depends on the texture, color, reflectivity, orientation and distance of
the surfaces under the sensor. Under controlled conditions, we use it to determine the position of points on a horizontal edge over a black background: Given the estimated pose of a point, the sensor moves along a line perpendicular to the edge. A proportional controller controls the position such that the reflected amount of light reaches a certain value. The final position, with a feature specific offset added, gives the position to be returned. Multiple observations of the same point vary ±0.02mm around their mean, which is due to the limited resolution of the manipulator's position.

This edge point location procedure is used to obtain more precise information on objects whose pose is known to within ca. 3 mm and 10 degrees, with object and feature sizes being around 10 to 100 mm. A more complicated algorithm locates rectangular blocks known to lie somewhere within a 100 by 100 mm square, using only the infrared sensors.

Vision system

A vision system using binary images locates and identifies objects lying on a plane. It uses global contour features such as area, circumference and main axes [OK87]. In addition to object type and coordinates, it also returns the degree of ambiguity in object positions: Symmetric objects may e.g. have a rotation ambiguity of ±180 degrees. This must be known to the ARGUS procedure which attempts to match observed objects to parts in the workspace model. The vision system is written in Modula-2 and runs on a PC compatible computer with a frame grabber card.

5.2.2 Software

MPW Shell: The Macintosh Programmer's Workshop (MPW) is an environment for developing applications in various programming languages. It provides a window front-end for applications that do not contain Macintosh window handling routines.

Prolog in its pure form represents predicate calculus statements and has a mechanism for combining them to prove given goal propositions. The extended form used for programming is a declarative language, in contrast to the usual procedural languages. Ideally, a Prolog program describes the solution of a problem rather than the path the solution takes. Prolog allows to store, retrieve and manipulate arbitrary data structures easily. For this reason, and since Prolog programs usually are interpreted, it is well suited for prototyping. If required, program structures familiar from procedural languages can be programmed in Prolog itself. The classic Prolog textbook is [CM84].

The Prolog version used in this project provides a very fast interpreter and allows to compile Prolog programs [Gmb88]. Most important, it can be linked to programs written in the C language under MPW. This facility was used to integrate the Prolog interpreter into the M-EANS system (written in the Ada language), and to make numerical routines written in C available.
5.2. Implementation environment

C is a programming language combining high level language features with (unprotected) access to the machine level. This makes it suited for writing run-time efficient programs, at the cost of implementation efficiency. We use C to implement matrix operations. Procedures for matrix inversion and eigenvalue computation are taken from [PFTV88].

Matlab started as a command driven interactive program for matrix analysis. Current versions include plotting facilities and the ability to execute external function files. The Matlab core is complemented by sets of such functions, called toolboxes, ranging from control theory to chemical analysis. We used Matlab to create a simple robot simulation environment and to develop the numerical procedures which then were implemented in Prolog. When working with ARGUS, mainly the graphic capabilities of Matlab are used, but it is also applied to verify numerical results obtained by ARGUS.

AML/2 is the programming language IBM provides for its manufacturing system [Cor86, NLT+86]. AML/2 is not just an enhanced robot programming language, but a high level language incorporating manipulator commands. It provides object oriented features, exception handling, and the ability to evaluate expressions at run time, as do Prolog or Lisp. A prototype object level system was written entirely in AML/2 [Mü87]. Since AML/2 programs are interpreted and execution is slow, the current ARGUS/M-EANS system uses AML/2 only for robot level procedures.

5.2.3 Prolog tools

Here we describe programming tools that were developed for ARGUS but can also be used in another context. In addition to the knowledge model (see section 4.2) and miscellaneous predicates realizing e.g. if - then - else structures, the following tools are of interest:

Program modularization

The Prolog language itself does not provide for, let alone enforce the modularization of large applications. Some minimal discipline is needed to group related procedures into the same source file and to minimize the use of procedures declared in other files. We encourage this by providing two utility procedures: One procedure allows to specify, at the beginning of each Prolog code file, the names of other files on which it depends. When consulting the file (i.e. loading it into the interpreter), the specified files will automatically be consulted first. This may happen recursively, while repeated reconsulting of the same file is avoided. A list of directory paths to be searched may be given, so that files are not restricted to a single working directory.

The second utility uses the consulting dependencies to create a cross reference listing of procedures: Given a file name, its procedures and the files it depends on are analyzed recursively. The analysis returns a cross reference listing of procedures (including clauses asserted at run time), a list of procedures that are never used, and a list of
undefined procedures. It still is possible for run time inconsistencies to occur, since procedures may be assembled and then defined or called at run time.

**Prolog debugger**

No useful debugger was provided with our Prolog implementation. Therefore, a debugger with most of the features of the classic DEC-10 Prolog implementation was realized.

**Matrix operations**

Operations on geometric data require matrix computations, which are cumbersome to implement in Prolog. They were therefore implemented in the C language and linked to the Prolog interpreter. In Prolog, operators were defined that allow a functional representation of mathematical expressions. An expression is evaluated by the Prolog clause

\[ \text{Result} \equiv \text{'eqs' expression}. \]

and its result is matched to the Prolog variable \( \text{Result} \). An expression consists of

- **Matrices**, represented by nested Prolog lists or by one of the following symbolic forms: \( \text{eye} \), \( \text{zeros} \), \( \text{inf} \)
- **Operators** from the set \( \{ +, -, *, /, \text{invtraf}, \text{transpose}, \text{eltimes}, \text{eldiv} \} \), the last two being element-wise operations. The \( \text{invtraf} \) operation inverts a homogeneous transform matrix more efficiently than \( \text{inv} \).
- **Functions**, that e.g. return the description vector or the Jacobian of a transform (and vice versa), or transform an uncertainty to another frame.

Expressions may contain arbitrarily nested operations and functions. They are evaluated by decomposing them recursively into elementary operations, checking for errors and calling numerical routines when necessary. Their result values are combined, working backwards until the original expression's value is obtained. During this process, time is saved by treating the symbolic forms of matrices specially, e.g. when the neutral element of an operation occurs. Some examples for expressions and their results are (The procedure \text{writeMAT} displays matrices in Matlab readable form):

?– A eqs [[2], [3], [4]] * [0.5, 1, 2], writeMAT(A).

\[
\begin{array}{ccc}
1.0 & 2.0 & 4.0 \\
1.5 & 3.0 & 6.0 \\
2.0 & 4.0 & 8.0 \\
\end{array}
\]
5.3. Limitations of the current implementation

There exist two implementations of the function structure outlined in Fig. 5.1. The first incorporates a dedicated planner, which works without explicit agent descriptions, and does not model uncertainties. It is interfaced to the M-EA\textsc{Ns} system and to the robot level, and is used to plan and execute operations with the real robot.

The second implementation incorporates the planner and uncertainty modeling described in chapter 4, but is only interfaced to the simulated robot. The uncertainty precondition checks and fixing strategies are not implemented. Primitive agents are
defined for the operations described in sections 5.1.4 and 5.2.1. This implementation has been used to verify the planning of acting and sensing operations separately.

In order to complete the system and to extend its generality, it is necessary to implement

- uncertainty precondition evaluation and fixing strategies, such that plans with acting and sensing operations can be generated fully automatically.
- the capability to execute a finished plan, using the real assembly system.
- more acting agents, incorporating different strategies for grasping and mating.
- more sensing agents, in particular for the detection of object features with a stationary or mobile 2D or 3D image sensor.
This chapter shows the application of the procedures developed in the last two chapters. They are used to plan acting and sensing operations for a pick and place task. First, the part descriptions and workspace initialization are shown. The computation of relative uncertainties is then demonstrated separately, since it is vital to most of the other functions. Next, the planner is applied, generating a plan while simulating the effects of the operations it considers. We then deliberately introduce pose errors into the finished plan, such that its execution shows how part poses are updated. The last two sections give examples for the planning and the integration of observations.

6.1 OBJECT AND WORKSPACE MODELING

Recall from section 4.3 that objects and their potential relations are represented by “types” and “ids”. The part types used in the following examples are shown in Fig. 6.1. An example for a part type definition was given in the preceding chapter. Using these part types, the assembly shown schematically in Fig. 6.2 was defined. The following examples are taken mainly from the description of the subassembly shown in Fig. 6.3. To begin with, each part of the assembly is assigned a unique id:

```cpp
insert partid with [
    id : 'SA_LC',
    type : largecyl,
    placeon : table, % no fixture
    placetraf : eye,
    graspinfo : largecylGrasp ].
```

```cpp
insert partid with [
    id : 'SA_SB',
    type : smallblock,
    placeon : table,
    placetraf : [-77.5,-35,0], % for centering
    graspinfo : smallblockGrasp ].
```
Figure 6.1: Example part types. The part types are called smallcyl, largecyl, smallblock, largeblock. The blocks and the large cylinder have holes in their top surface. The part frames of the two blocks are located at the circled vertices.

Figure 6.2: Example assembly with part ids. Individual parts of the same type must have different id’s. Abbreviations: A: Assembly, SA: Subassembly, LC: Large Cylinder, SB: Small Block, etc. Loose joints are indicated by bold lines.
6.1. Object and workspace modeling

% etc... for the other ids (9 in total)

For the above partid 'SA_SB', the value of the placenp attribute, which is the transform from the part frame to the frame of a storage location or fixture, ensures that the part is centered at the storage location. The part ids of the same type here share the same grasp information. In general, they may have different and multiple grasp points. One grasp point is described by e.g.

```plaintext
insert graspinfo with [
  id : largecylGrasp,
  gripper : 'SGR1000', % physical gripper
  grasppnt : [0,0,12,0], % grasp point
  graspclos : 30.0, % nominal width
  graspopen : 36.0, % width before grasp
  graspsorc : 20.0 ]. % force when grasping
```

% etc... at least one for each part type

When a partid is inserted, its graspinfo is automatically retrieved and used to create a mateid, e.g. graspPt_6...for-SA...LC, representing the potential joint between the gripper and the part.

The relations between parts are described by matetypes and mateids. Types describe the common properties of different joints: There is a type for the joints established by grasping with a parallel gripper, and there are types for loose or fixed prismatic and cylindrical joints. Since the types provide default information for their descendant classes, values must be given for all attributes. The values eye and minunc are symbolic matrices denoting the identity matrix and a minimum uncertainty matrix. Of interest are the max_joint_unc attributes expressing joint tolerances. The values ud and uq are symbolic scalars denoting maximal (the staff of the letter “d” points upwards) and minimal (the staff of “q” points downwards) uncertainty values. They will be replaced by numerical values that allow computations without numerical difficulties.

The properties of a parallel gripper, whose joint frame lies between the gripper's fingertips and has its y axis aligned with the direction in which the fingers move, are expressed by

```plaintext
insert matetype with [
  type : paralgrip,
  jointtraf : eye,
  joint_unc : [ud,uq,ud,uq,ud,uq],
  top2joint : eye,
  top2joint_unc : minunc,
  joint2bot : eye,
  joint2bot_unc : minunc,
  uncbefore : minunc,
  max_joint_unc : [ud,uq,ud,uq,ud,uq],
  strength : fixed ].
```
Chapter 6. Examples

Figure 6.3: Example subassembly part and joint types and ids. Part types (t:) and ids (i:) are shown to the left, in the same order as the part frames. Joint types and ids are shown to the right. The joint frame of tempTableMateId1_for_SA_SB and camera_to_SA_SB is indicated by the shaded frame. The other joint frames coincide with the frames of the joint's top objects.

The maximum joint uncertainty states that, after gripping, the linear uncertainty in the y direction and the angular uncertainties about the x and z axes are minimal, and that the other uncertainties are maximal. The joint uncertainty, which here only is a default value for the joint uncertainties of the descendant mates, is set to the maximum joint uncertainty.

Another frequently used matetype describes the planar joint established when a part lies on the table in the robot's workspace:

```
insert matetype with [
    type       : genericplace,
    joint_unc  : [10,10,uq,uq,uq,0.1],
    max_jointunc : [ud,ud,uq,uq,uq,ud],
    strength    : loose,
    ...
]
```

The joint frame's z axis is normal to the plane. The maximum joint uncertainty expresses that the part cannot move in the z direction or rotate about the x or y axes. Its joint uncertainty has finite values for the other directions, since the part is assumed to lie within the finite workspace. The values given here have been chosen quite small, since they are also used in the uncertainty computation examples of the next section. The linear uncertainty value of 10 means that the probability of the absolute error being smaller than $3\sqrt{10}$mm $\approx 9.5$mm is virtually 1.
The maximum joint uncertainty of all fixed joints is minimal. Loose joints may have values as in

```
insert matetype with [ % loose cylindrical
type : pih_circle_loose, % pih:PegInHole
joint_unc : minunc,
max_joint_unc : [0.02,0.02,0.02,
               0.002,0.002,ud],
strength : loose
...remainder as in paral grip
```

% similar for pih_circle_fixed, pih_square_fixed (with % minimal max_joint_unc), and pih_square_loose

When inserting a mateid, it is only required to specify the attributes which differ from the defaults given in its type:

```
insert mateid with [ id : 'SA_LC_2_SA_SB' ,
type : pih_circle_fixed,
topid : 'SA_LC',
botid : 'SA_SB',
joint2bot : [-25,-25,-5]. % descr. vector
```

% etc... one mateid for each potential joint (8 in total)

Observation joints are described by a feature specific type and an object specific id. One feature is the center of the object contour observed from above. Its pose provides information on the x and y translation and the z rotation only. This is expressed in the joint_unc. A default value for the sensing accuracy is given in top2joint_unc. Its actual value, which may depend on the observation, will be entered in its obsinst. The obsid states, for each part, the relation from the contour center to the part frame.

```
insert obstype with [ type : vs_center_of_gravity,
jointtraf : eye,
joint_unc : [uq,uq,ud,ud,ud,uq], % feature
top2joint_unc : [ 1, 1,uq,uq,uq,0.03], % default
joint2bot_unc : none,
strength : fragile ].
```

```
insert obsid with [ id : camera_to_SA_SB ,
type : vs_center_of_gravity,
topid : camera ,
```
% etc.... one obsid for each part (9 in total)

This completes the description of the parts and their assembly. The description of the robot, the gripper and fixed objects in the workspace is similar. For other obatypes and -ids, see section 6.5.

Now we can model a state of the workspace by creating partinsts and mateinsts. They state that "there exists an object X at location Y". The location information is either known a priori or determined by sensing. For example, we wish to instantiate a part with the id 'SA_SB' located at the position $[-300,100,-400]$ and rotated 30 degrees about the z axis of the part frame. This normally is done automatically, and results in the following insert operations: First, the part is instantiated. Its type is retrieved and used to generate a unique label for the inst attribute, e.g. as in

\begin{verbatim}
insert partinst with [ 
    inst : smallblock_2,
    id : 'SA_SB',
    orderid : unused ] .
\end{verbatim}

Since the location is at an arbitrary place on the table, a temporary mateid to the world frame is generated. It contains the given location (inverted) and the part's place transformation (retrieved from the part description).

\begin{verbatim}
LocT eqs invtraf desc2traf([-300,100,-400,0,0,pi/6]),
retrieve partid with [id: 'SA_SB', placetraf: PlaceTraf ],
insert mateid with [ 
    id : tempTableMateId2_for_SA_SB,
    type : genericplace,
    topid : 'SA_SB',
    botid : world,
    top2joint : PlaceTraf,
    joint2bot : LocT,
    temp : true ].
\end{verbatim}

Finally, a mateinst is created, relating the partinsts and the mateid.

\begin{verbatim}
insert mateinst with [ 
    inst : tempTableMateId2_for_SA_SB_inst1,
    id : tempTableMateId2_for_SA_SB,
    topinst : smallblock_2,
    botinst : world ].
\end{verbatim}

The same is done for the robot, table and all parts entering the workspace.
6.2 UNCERTAINTY COMPUTATION

Planning and operations modeling very often require the computation of relative un­
certainties, which therefore is demonstrated separately. Let the workspace have the
relations shown in Fig. 6.4. The relations between parts have minimum uncertainty,
the observation and placement relations have the default uncertainties defined in the
types vs.center_of.gravity and genericplace. We first extract the uncertainties
of selected relations. The call to the procedure relUnc is shown with only its input
arguments i.e. the two nodes and the list of nodes to be ignored. The computation
time and the resulting linear and angular uncertainties are displayed. The uncertainty
ellipsoids all have the same orientation.

The relative uncertainty of largeblock_1 with respect to world, ignoring the camera,
should result in simply the default uncertainty of the matetype placetraf:

\[
\text{relUnc}(\text{world}, \text{largeblock}_1, [\text{camera}]), \quad \% \text{two nodes} \\
\text{lin}_\text{unc} = [10.0, 10.0, 3e-08] \\
\text{ang}_\text{unc} = [3e-08, 0.1] \\
\% \text{0.7 sec}
\]

The uncertainty of the observation of largeblock_1, which is the observation uncer­
tainty of the obstype vs.center_of.gravity, is obtained by ignoring the world and
smallcyl_1 nodes.

\[
\text{relUnc}(\text{camera}, \text{largeblock}_1, [\text{smallcyl}_1, \text{world}]). \\
\text{lin}_\text{unc} = [1.00505, 1.00102, 1e+08] \\
\text{ang}_\text{unc} = [1e+08, 0.03] \\
\% 1.8 sec
\]

If the observation of smallcyl_1 is not ignored, we obtain for the same two nodes

\[
\text{relUnc}(\text{camera}, \text{largeblock}_1, [\text{world}]). \\
\text{lin}_\text{unc} = [0.495199, 0.497126, 5e+07] \\
\text{ang}_\text{unc} = [0.620319, 0.0198298, 0.000480579] \\
\% 5.4 sec
\]
Figure 6.5: Uncertainty visualization. Observation, placement and combined uncertainties. The vertical lines are parallel to the z axis in three dimensions. All other lines lie in the xy-plane. Their rotation about the z axis is arbitrary, since (in this particular case) the uncertainties in x and y direction are equal. Shaded lines show angular uncertainties.

The x and y uncertainties now are halved, since both observations have the same accuracy. In the z direction there still is essentially no information. The angular uncertainties about the x and y axes are reduced significantly, although each observation by itself provides no information. This reduction is due to the fact that the distance between the largeblock_1 and smallcyl_1 frames as well as their projections onto the xy plane are known, constraining the orientation of the assembly.

The complete information between the largeblock_1 and world frames is contained in the uncertainties found by

\[
\text{relUnc(world, largeblock_1, [])}.
\]

\[
\text{lin_unc = [0.497205 0.494843 3e-08 ]};
\]

\[
\text{ang_unc = [3e-08 3e-08 0.000304845 ]};
\]

The x and y uncertainties still are determined essentially by the observations, and the small values of the other elements are due to the constraints of the planar support. If the the path through smallblock_1 is ignored, the uncertainty increases only insignificantly, since the location of the camera relative to the world frame is already very precise. The computation time, however, is reduced to one third.

For a visualization, see Fig. 6.5. It shows the uncertainties of the placement of largeblock_1 and of one observation, as well as the combined uncertainty. The numerical values are essentially as in the above example.

6.3 PLANNING AND OPERATIONS MODELING

Let the workspace state be as in the last example (Fig. 6.4), but without the observation of smallcyl_1. The uncertainties of the parts with respect to the world frame are
6.3. Planning and operations modeling

lin_largeblock_1_unc = [0.931391 0.930719 3e-08 ];
ang_largeblock_1_unc = [3e-08 3e-08 0.0014303 ];

lin_smallblock_1_unc = [0.311854 0.31278 3e-08 ];
ang_smallblock_1_unc = [3e-08 3e-08 0.00432694];

The observation of smallblock_1 has been assumed to be more precise than the other one, such that the linear uncertainty of smallblock_1 is smaller than that of largeblock_1.

Starting at this initial state, we wish to mate smallblock_1 to largeblock_1. A complete plan is generated by the command

plan puton( smallblock_1, largeblock_1 ).

smallblock_1 will be grasped, moved and mated to largeblock_1. The resulting qualitative changes to the network of relations have already been shown in section 4.1.6 and Fig. 4.8. Since we wish to discuss some intermediate states that are generated during the planning process, we shall create the plan stepwise. The first step is

plan grasp( smallblock_1 ).

Note that we chose to grasp not smallblock_1, but largecyl_1 instead. Since the two are affixed to one another, the subsequent move and mate operations will not release largecyl_1 and grasp smallblock_1, but keep on grasping largecyl_1.

An arbitrary amount of tracing information may be output. Some of it is reproduced here, with comments added. Transform and uncertainty matrices are replaced by empty matrices [ .. ].

% Is largecyl_1 already being grasped by some gripper?
test lpredc holding(largecyl_1) of already_grasping(largecyl_1)
% Find a gripper that can grasp it.
test lpredc canGrasp(largecyl_1) of basic_grasp(largecyl_1)
% Expand the found agent.
expanded basic_grasp
  % Is the gripper grasping something else?
test lpredc holding(sgripper,X), not X = largecyl_1)
    of free_gripper_grasp(sgripper,largecyl_1)
  % Since it does not, the next agent can be expanded.
test lpredc not holding(sgripper,X)
    of direct_grasp(sgripper,largecyl_1)
expanded direct_grasp
  % The next agent has no logical precondition, expand it.
test lpredc default
    of move_to_grasp(sgripper,largecyl_1,graspPt_6_for_SA_LC)
expanded move_to_grasp
The expansion does a command to move the robot and set the gripper position. The values shown are robot coordinates. simulated safemove_grpos to [-288.636, 130.117, -45.6, 0, -0, 0.5236] with GrPos 36.0

Compute the two parts of the relative uncertainty between the gripper and largecyl_1.

```plaintext
did relUnc(world, sgripper, [camera])
did relUnc(world, largecyl_1, [endofarm])
```

There is no logical precondition for the approach operation.

```plaintext
test lprecd default of blind_approach
```

Its uncertainty precondition uses the two above uncertainties.

```plaintext
% It is not implemented and therefore always succeeds.
```

test uprecd check_blind_approach_uprecd

expanded blind_approach

The expansion does a guarded move and closes the gripper. simulated gmate by [0.0, 0.0, -30.0, 0.0, 0.0, 0.0]

The gripper width is set to the nominal width of largecyl_1.

```plaintext
gripperpos 30.0
```

used by gripperforce 20.0

Update the joint transformations. They will not be changed, since the ideal positions are used in the simulation.

```plaintext
did modify tempTableMateId1_for_SA_SB_inst1 with jointtraf eye
did modify SA_LC_to_SA_SB_inst1 with jointtraf eye
did modify graspPt_6_for_SA_LC_inst1 with jointtraf eye
```

Update the physical joint uncertainties, reflecting the constraints of the joints around the new kinematic loop.

```plaintext
% For each joint, the diagonal of the L matrix (section 4.1.3) is shown. Its entries determine whether the uncertainties along the 6 axes are set to the old, the maximum, or the maximum joint uncertainty (mju, which usually is small)
```

```plaintext
did modify tempTableMateId1_for_SA_SB_inst1 with joint_unc [...] 
diag(L) = [ 0.5 1.0 0.0 0.0 0.0 1.0 ];
% New unc is: x:old y:max z:mju xr:mju yr:mju zr:max
% I.e: the placement joint may have complied in y and zr dirs.
% The corresponding uncertainties are maximized.
```

```plaintext
did modify SA_LC_to_SA_SB_inst1 with joint_unc [...] 
diag(L) = [ 0.0 0.0 0.0 0.0 0.0 0.0 ];
% All new unc. are mju: the joint fully constrains itself.
```

```plaintext
did modify graspPt_6_for_SA_LC_inst1 with joint_unc [...]```
diag(L) = [ 0.5 0.0 1.0 0.0 1.0 0.0 ];

% New unc is: x:old y:mju z:max xr:mju yr:max zr:mju
% The grasp relation is constrained in the y, xr, and zr dirs.

% Update the uncertainty of the observation of largecyl_1, using
% the L matrix of largecyl_1. Since the observation parameters
% cannot be modified, the observation is replaced by a new one:
did_insertSimpleObs camera largecyl_1

% The complete plan is:

grasp(largecyl_1)
grasp(sgripper,largecyl_1)
findJointId(sgripper,largecyl_1,graspPt_6_for_SA_LC,largecyl_1)
move_to_grasp(sgripper,largecyl_1,graspPt_6_for_SA_LC)
  getApproachPt(largecyl_1,graspPt_6_for_SA_LC,[..])
  getGripSettingBeforeGrasp(graspPt_6_for_SA_LC,36.0)
  gross_move_grpos(sgripper,[..],36.0)
calcUncForApproach(sgripper,largecyl_1,[..],[..])
approach(sgripper,largecyl_1,graspPt_6_for_SA_LC,[..],[..])
  getApproachDMove(graspPt_6_for_SA_LC,[..])
  cg_dmove(sgripper,[..])
activateGripper(graspPt_6_for_SA_LC)
create_phys_joint(sgripper,largecyl_1,graspPt_6_for_SA_LC)

The state of the workspace model now is as in Fig. 6.6, where the uncertainties
of both blocks with respect to the world frame are visualized. The uncertainty of
largeblock_1 is unchanged. The uncertainty of smallblock_1 originally was similar
to largeblock_1's. Now its linear part is reduced in the direction of grasping, and
unchanged in the other. Its angular uncertainty about the z axis has also been reduced:

lin_smallblock_1_unc = [0.309118 0.00357704 3e-08 ];
ang_smallblock_1_unc = [2.41e-08 3e-08 1.2e-07 ];

These uncertainties result from the combination of the uncertainties of the following
joints (all uncertainty ellipsoids are aligned with the grasp point frame). The grasp
joint provides information on y, \( \phi_z \), \( \phi_z \), the placement joint on x, z, \( \phi_z \), \( \phi_y \), and the observation on x only:

lin_graspPt_6_for_SA_LC_inst1_unc = [1e+08 0.000 1e+08 ];
ang_graspPt_6_for_SA_LC_inst1_unc = [3.7e-08 1e+08 3e-08 ];
lin_tempTableMateld1_for_SA_SB_inst1_unc = [9.9994 1e+08 3e-08 ];
ang_tempTableMateld1_for_SA_SB_inst1_unc = [3e-08 3e-08 1e+08 ];
lin_sObs_1_camera_largecyl_1_unc = [0.2948 1e+08 1e+08 ];
ang_sObs_1_camera_largecyl_1_unc = [1e+08 1e+08 1e+08 ];
The next step being planned is:

plan puton( smallblock_1, largeblock_1 ).

In place of another list of tracing information, we just give its summary:

- If the goal does not happen to be satisfied already, check whether there is a free joint for smallblock_1 on largeblock_1. Then check whether smallblock_1 is already being grasped. Since it is being grasped indirectly, via largecyl_1, no grasp action is required.
- Unmate smallblock_1 from its support: Save the information in the observation of largecyl_1 by deleting it and creating an observation relative to the gripper. Since the new observation is parallel to the grasp joint, its information is added to the grasp joint’s. Save the placement information of smallblock_1 in an observation relative to the gripper.
- Simulate the getaway move.

The remaining trace is similar to that of the grasp operation, checking preconditions and updating uncertainties. The complete plan is:

puton(smallblock_1,largeblock_1)
findJointId(smallblock_1,largeblock_1,SA_SB_to_A LB,largeblock_1)
puton(smallblock_1,largeblock_1,SA_SB_to_A LB)
unmate(smallblock_1)
   matedInstId(smallblock_1,world,tempTableMateId1_for_SA_SB_in..
delete_phys_joint(smallblock_1,world,tempTableMateId1_for_SA..
getaway(smallblock_1,world,tempTableMateId1_for_SA_SB)
getGetawayDMove(tempTableMateId1_for_SA_SB,invtraf ..)
gross_dmove(smallblock_1,invtraf ..)
mate(smallblock_1,largeblock_1,SA_SB_to_A_LB)
move_to_mate(smallblock_1,largeblock_1,SA_SB_to_A_LB)
getApproachPt(largeblock_1,SA_SB_to_A_LB, ..)
gross_move(smallblock_1, ..)
calcUncForApproach(smallblock_1,largeblock_1, .., ..)
approach(smallblock_1,largeblock_1,SA_SB_to_A_LB, .., ..)
getApproachDMove(SA_SB_to_A_LB, ..)
cg_dmove(smallblock_1, ..)
create_phys_joint(smallblock_1,largeblock_1,SA_SB_to_A_LB)

The state of the workspace model is as in Fig. 6.7. The uncertainties now are

\[
\begin{align*}
\text{lin}_{\text{largeblock}_1} &= [0.314224, 0.003280, 3e-08] \\
\text{ang}_{\text{largeblock}_1} &= [2.41e-08, 2.5e-08] \\
\text{lin}_{\text{smallblock}_1} &= [0.314114, 0.002843, 1.9326e-06] \\
\text{ang}_{\text{smallblock}_1} &= [3.68e-08, 4.38e-08, 1.2e-07]
\end{align*}
\]

Since the parts now are affixed to each other, with small relative uncertainty, the uncertainties are essentially the same. The uncertainty of \text{largeblock}_1 is reduced. It is determined, in one linear direction, by the original observation information on \text{smallblock}_1 (which has been merged into the grasp joint), and in the other by the accuracy of the grasp of \text{smallblock}_1. The original observation information on \text{largeblock}_1 has been completely invalidated, since the joint between the two blocks may have forced \text{largeblock}_1 to comply in all directions.

Finally, \text{largecyL1} is released and the gripper is retracted by
plan ungrasp( sgripper ).

- Check whether the gripper is holding something, then simulate the opening of the gripper fingers.
- The observation between the gripper and smallblock_1 (which contains some of its original placement information) must be deleted. Save its information by propagating it to a relation between largecyl_1 and smallblock_1 and adding it to the existing joint's information. Save the information of the deleted grasp joint in an observation of largecyl_1 relative to the world node.
- Simulate the getaway move.

The uncertainties now are

\[
\begin{align*}
\text{lin}_{\text{largeblock}_1}\text{unc} &= [0.325047, 0.0041085, 3e-08]; \\
\text{ang}_{\text{largeblock}_1}\text{unc} &= [2.55e-08, 3e-08, 1.7e-07]; \\
\text{lin}_{\text{smallblock}_1}\text{unc} &= [0.323898, 0.0037613, 1.935e-06]; \\
\text{ang}_{\text{smallblock}_1}\text{unc} &= [4.2e-08, 6e-08, 1.4e-07];
\end{align*}
\]

They have increased, but only slightly, since the uncertainty of the gripper pose was added to the grasp uncertainty before propagating it into the new observation of largecyl_1.

### 6.4 OPERATIONS MODELING: POSE UPDATE

When modeling during planning, the changes of transforms constrained by mating are zero, since the planner retrieves its information from the same model which it then modifies. In order to demonstrate the update of poses, we shall take a finished plan, introduce pose errors, and execute it. From the plan for grasping smallblock_1, as generated earlier, we extract the motion and model update commands, which are

- \texttt{gross\_move\_grpos(sgripper, NominalTransform ,36.0).}
- \texttt{cg\_dmove(sgripper, ApproachMove).}
- \texttt{activateGripper(graspPt\_6\_for\_SA\_LC).}
- \texttt{create\_phys\_joint(sgripper, largecyl\_1, graspPt\_6\_for\_SA\_LC).}

The workspace is set to its initial state and the commands are executed. If a linear error of \([-2, +2, -2]\) along the x, y, and z axes is added to the NominalTransform given to the \texttt{gross\_move\_grpos} command, then the tracing messages of interest are

- \texttt{did modify tempTableMateId1\_for\_SA\_SB\_inst1 with jointtraf [ 0.0, 2.0, 0.0, 0.0, 0.0, 0.0]}
- \texttt{did modify SA\_LC\_to\_SA\_SB\_inst1 with jointtraf [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]}
- \texttt{did modify graspPt\_6\_for\_SA\_LC\_inst3 with jointtraf [ -2.0, 0.0, -2.0, 0.0, 0.0, 0.0]}
6.5 AUTOMATIC SENSING

In section 4.4.5, a simple heuristic procedure for choosing contour features was described. Here it is incorporated into a sensing agent which, given a part instance and the minimum translation information needed about the part, selects and executes observations. It uses, in addition to the vision system feature introduced earlier, features corresponding to object edges which may be observed by a camera or a distance sensor. The feature description contains the feature specific uncertainty between the observed and actual feature, as well as a default observation accuracy:

```plaintext
insert obstype with [ 
    type : edge,
    jointtraf : eye,
    joint_unc : [ud, uq,ud,ud,ud,ud], % feat specific
    top2joint_unc : [uq,0.02,uq,uq,uq,uq], % obs specific
    joint2bot_unc : none,
    strength : fragile ].
```

The object used in this example is shown in Fig. 6.8. One of its edge features is described by

```plaintext
insert obsid with [ 
    id : wedgedge1,
    type : edge,
]
```

Figure 6.8: Observation example object. Left: Perspective view of the wedge object and the frames of its edge features. Top right: View from above with edge numbers and angle of maximum needed information. Bottom right: Linear uncertainty before observing, required maximum uncertainty and final uncertainty after observing.

The error in y direction was absorbed by the placement transform, i.e. the part was pushed in y direction. The other errors were absorbed by the grasp relation, i.e. the complete grasp transform now consists of its nominal transform plus the error.
topid : dummysensor,    % > 1 sensors allowed
botid : wedge,
joint2bot : [10,0,-10,pi],    % feature pose
featdata : 20,    % edge length
temp : false ].

The topid attribute is not restricted to a specific sensor, since different sensors may be used to locate an edge.

In the example, the needed information is specified by giving a diagonal matrix in the part frame and rotating it around the z axis. The steps taken during planning are explained in the following trace

Angle = 120,
NeededInfo eqs trafinfo( diagmat([2,0.05,0,0,0,0]),
     desc2traf([0,0,0,0,0,Angle]) ),
plan observe_pose(wedge_1,NeededInfo).

% Is the vision system (vs) accurate enough?
test lprecid can_provide_info(vs,wedge_1,[..]) of vs_obs_pose
% It is not, but the infrared sensors (irs) are
test lprecid can_provide_info(irs,wedge_1,[..]) of irs_obs_pose
% The uncertainty precondition is not implemented and succeeds.
test uprecid check_irs_obs_pose_uprecid
expanded irs_obs_pose

% The difference between the available and required information
% gives the direction in which most information is needed.
% This direction determines the edge(s) that will be observed:
selectEdgeObs gives obolist: [wedgedge2 ]

% The gripper is closed, the finger with the sensor is moved
% to the expected edge location and measures it.
gripperpos 7
simulated safemove to [-471.092,32.4067,-70.6,0.0,-0.0,-2.6779]
simulated obsEdge of wedge_1 at edge Pos [-460.0,10.0018,-390..

The linear uncertainty is reduced as desired. A qualitative display of the involved uncertainties is in Fig. 6.8. If the angle in which the needed information is maximal is varied, the edges selected for observation will change:

<table>
<thead>
<tr>
<th>Angle (degrees)</th>
<th>Observed edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>3, 4</td>
</tr>
<tr>
<td>80</td>
<td>3</td>
</tr>
<tr>
<td>120</td>
<td>2</td>
</tr>
<tr>
<td>160</td>
<td>4</td>
</tr>
</tbody>
</table>
6.6 OBSERVATION INTEGRATION

A network of relations has its transforms updated when an observation is integrated. For the example, the network of Fig. 6.4 is used again. A new observation between the camera and smallblock_1 is added. The observed center of the block is displaced, relative to the expected center, by 10 mm in x, and 5 mm in y direction. See Fig. 6.9. The observation uncertainty is, along the x axis, approximately equal to the a priori uncertainty. Along the y axis, it is ca. one fourth of the a priori uncertainty. Therefore, discrepancies along the x axis should be distributed evenly between the observation and the part’s placement relation. Discrepancies along the y axis should cause the change in the placement relation to be four times larger than in the new observation. The insertion and integration of the new observation are done by executing

\[
\text{create_obs_joint} \left( \text{camera, smallblock}_1, \text{camera_to_SA_SB}, \text{ObsT}, \text{ObsU} \right).
\]

where ObsT, ObsU are the observation transform and uncertainty. The recursive description of the affected part of the relations network is, if available, retrieved from memory, otherwise it is computed. As it is updated, the relations are updated in the workspace model as well. Among all the changes, the following are of significant magnitude:

<table>
<thead>
<tr>
<th>Relation:</th>
<th>Change: x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>world --&gt; smallblock_1</td>
<td>[ 4.769,</td>
<td>3.921, 0.0 ]</td>
<td></td>
</tr>
<tr>
<td>camera --&gt; smallblock_1</td>
<td>[ -5.254,</td>
<td>-1.098, 0.02 ]</td>
<td></td>
</tr>
<tr>
<td>camera --&gt; largecyl_1</td>
<td>[ 4.698,</td>
<td>3.804, 0.079 ]</td>
<td></td>
</tr>
</tbody>
</table>

The first two show how the error is distributed over the expected pose and the observation. The third shows the update of the earlier observation of largecyl_1, which is affixed to smallblock_1.

\[^1\text{Here, and in the remainder of the example, displacements and uncertainties are described in frames parallel to that of smallblock_1.}\]
In this case, the network was reduced by merging and compounding alone. If a relation (with zero information) is introduced between e.g. smallblock.1 and largeblock.1, then the network becomes irreducible: After merging and compounding a few relations, the DIM of the simplified network must be assembled and inverted. The resulting updates are not shown since they are the same as above.

When angular errors are introduced, the updates are less easy to interpret, which is why we give no further examples. For large angular errors, the linearity assumption for transforming errors and variances does not hold [SC86].
Chapter 7

CONCLUSIONS AND FURTHER WORK

7.1 CONCLUSIONS

The experience made by robot programmers is exemplified by the following quote from [LLLM84]:

The discrepancy between our early previsions and the actual investment in the LM system has been the consequence of several initial misjudgments:

- We underestimated the difficulty of implementing a reliable robot programming system.
- We overestimated our understanding of robot programming [...].

The author shares these feelings. The realization of a working assembly system required a large amount of work not immediately related to the theoretical aspects of planning and uncertainty manipulation. However, it created practical problems that motivated the investigation of relevant theoretical topics. Since theory tended to drift away into its own self-contained universe, it had to be pruned repeatedly to remain applicable to the non-ideal world of robotic assembly.

The result is twofold:

- A working system that has repeatedly and reliably demonstrated the simultaneous assembly of various products requiring small tolerances, and whose parts arrive at arbitrary times and locations.
- It has been demonstrated how geometric uncertainties can be modeled explicitly, and how they may be used when planning acting and sensing operations. A complete object level planning system will require more work.

In contrast to most related systems that take uncertainties into account [SC86, DW88, PTP88, MA89, HK90a, SL91], ARGUS allows to compute uncertainties and to model the effects of acting and sensing in arbitrary networks of relations.

The correctness of the modeled uncertainties, from the mathematical point of view, is limited by a number of factors:
• It is assumed that errors are small, such that the linearity assumptions implicit in the procedures for manipulating uncertainties hold.
• The update procedures for creating and deleting relations rely on a set of assumptions and use approximations and simplifications.
• Tolerances are modeled as uncertainties, and some model parameters are chosen by an educated guess.

However, the goal is not to have a perfect workspace model, but to have a basis for planning. The model provides useful information about the magnitude and – most important – the direction of possible pose errors. This suffices to reduce the search space in mating strategies or when interpreting sensor data, greatly reducing their execution time.

What is the practical use of the ARGUS system? Task level systems in general are still far from being applicable in industry, and so is ARGUS. But it does, on the one hand, provide a framework for research on the planner and agent level. On the other hand, a more practical application would be to use its uncertainty handling capabilities to assist the offline planning and testing of robot level programs: In an interactive system, a programmer would be informed about the need for reducing uncertainty actively or passively, and would be provided with suggestions. If the overall process is not to suffer, then of course the acquisition of model data must be automated.

7.2 FURTHER WORK

The functions implemented in the ARGUS system form a basis for research into the following areas, each of which provides work for quite a few man-years.

7.2.1 Uncertainty manipulation

The procedures for manipulating uncertainties have been shown to work in principle, but more testing is necessary to determine the exact causes and magnitudes of numerical and linearization errors.

Regarding the theoretical basis, the analogy to screw theory deserves to be pursued further. The modeling of operations could be improved by further study and application of the theory of kinetostatics of mechanisms [Phi84].

7.2.2 Plan fixing

Recall that, when an uncertainty precondition is not satisfied, uncertainty reducing operations must be introduced into the plan. The following problems arise:

• For a pair of objects whose relative uncertainty must be reduced, it may (in each direction) be necessary to observe either one or the other or both objects. If both have to be observed, a mutual dependency of the observations exists, which might be taken into account (see Fig. 7.1).
7.2. Further work

Figure 7.1: When to reduce uncertainties? At each step of the sequence of operations shown, uncertainty reducing operations (grasping, pushing, observing) may be inserted. What is an optimal strategy for selecting them so that the mate operation will succeed?

- When a fixing operation is inserted into a plan, its interference with other operations must be checked: On the one hand, it might invalidate their preconditions, on the other hand it might make other sensor applications redundant.
- A fix may require another fix, and replanning after fixing may again require fixes. This procedure is not guaranteed to terminate. A planner/fixer must be able to detect when a task cannot be accomplished and whether it runs in an endless loop.

7.2.3 Global error recovery

The operations planner could be enabled to handle the following problem: Agents are assumed to include strategies for detecting and handling errors locally. If an acting agent fails to execute its operation, or if a sensing agent cannot match its observation to expectations, then the error must be handled by the planner. Depending on the failure and on the available sensors, it may have to try other agents or check the validity of the workspace model.

7.2.4 Versatile agents

It is desirable to have versatile agents, such that the load on operations planning and coordination is reduced. For example, a human may pick up a part arbitrarily and then reorient it as is needed for mating. Then the mating operation does not constrain the grasp operation, thanks to the manipulative ability of our fingers. But if reorientation takes too much time, the overall process may not be optimal.

Smarter agents require hardware developments, the use of tactile, force and visual sensors, and strategies for using them to control the actuators. They are an inviting
field for the application of behavior based planning, neural networks, fuzzy logic and the like.

7.2.5 Sensor planning

The problem to be solved here is: What is the optimal (usually with respect to time) combination of observations that provides some required information, given the current constraints on sensor applicability? One solution is to express sensing strategies by a rule-based system and to verify them by simulation. Stating rules about the acquisition of either purely linear or angular information is relatively straightforward, but reasoning simultaneously about both will be more difficult.

7.2.6 Data acquisition and learning

Planning involves a wealth of data: General planning knowledge, planning knowledge for specific classes of tasks and for specific agents, object models, sensor and actuator positions, control parameters and sensor specific parameters. Acquiring this information is costly, and it must be updated as parameters and tasks change. This knowledge acquisition bottleneck motivates learning mechanisms, which should partition the domains of learning, from the robot retraining problem [Seg88] down to the estimation of sensor value offsets.

Some workspace model data can be generated from CAD models, e.g. suitable grippers, grasp points, features for observation by different sensors, joint types and tolerances. The uncertainties used in uncertainty preconditions for deciding the applicability of an agent could be determined either from CAD models or by practical experiments.

Directly related to the maintenance of the network of uncertain relations is the problem of calibrating the robot and the sensor locations: Note that any observation provides information not just on the pose of the observed objects, but also on the pose of the sensor, and possibly the robot. Therefore, suitable observation strategies using the uncertainty model implemented in ARGUS could be applied to the calibration problem.

7.2.7 Considering the broader context

The above problems pose interesting challenges. It is crucial that the scientist and engineer, when responding to the challenges, also consider the context and implication of her or his work: The automation of industrial production has often provoked discussions, since its blessings and curses are unevenly distributed over the populace and over time.

The positive and negative aspects of automation should be weighted against one another, in order to decide what level of automation is appropriate in a specific context. For successful automation projects, positive aspects (on the corporate level) include an increase of production quantity and quality, and the elimination of monotonous or
dangerous jobs. However, it must be noted that the monotony problem is a direct result of the tayloristic approach of job segmentation.

On the level of the whole national economy, negative effects on the structure of the work (and consumer) force, resulting from ever more automated production, are being discussed. Little consideration is given to global economy and resource management: Production automation replaces labour intensive processes by energy intensive processes, which, ironically enough, runs contrary to the trends of world population and energy availability.

The problem seldom addressed is whether the "pursuit of happiness" really must be realized by ever increasing production, consumption and throwing away of material goods. It has been noted that an "optimal quality of life" may be attained with a limited gross national product, since with the GNP exceeding a certain limit, the negative effects of its growth outweigh any positive ones.

The last question is where this limit should be placed, that is, how far we are willing to take the needs of the rest of this world and of our children into account.
The syntax of the definition of a knowledge model and of operations is described in EBNF. Terminal symbols are written as 'keyword' and nonterminal symbols as Nonterminal-Thing. Nonterminal symbols representing names of classes or attributes are written as Some-Name. Informal descriptions are preceded by the word "Note".

The symbols { } [ ] | ::= are meta-symbols belonging to the EBNF formalism.

Appendix A

KNOWLEDGE MODEL SYNTAX

Class-definition ::= 
    Class | Superclass | Type-Definition | Rule-Procedure

Class ::= 
    'class' Class-Name
    'attributes' Attribute-Value-Definition
    'identifier' Attribute-Name
    [ 'is_a' Attribute-Name ]
    [ 'rules' Rule-List ] '

Superclass ::= 
    'superclass' Superclass-Name 'contains'
    '[ ' Class-Name { ',' Class-Name } '] '

Attribute-Value-Definition ::= 
    '[ ' Att-Val-Def-Entry { ',', Att-Val-Def-Entry } ']'

Att-Val-Def-Entry ::= 
    Attribute-Name ':' Type
    [ ':' '[' Characteristic { ',', Characteristic } ']' ]

Type ::= 
    Class-Name | Simple-Type | User-Type

Characteristic ::= 
    'unique' | 'required' | 'unchangeable' | 'multivalued'

Simple-Type ::= 
    'atom_t' | 'atomic_t' | 'integer_t' | 'real_t' | 'numeric_t' | 'struct_t'
Appendix A. Knowledge model syntax

User-Type ::= 
   Type-Name

Type-Definition ::= 
   Type-Name (' Input-Argument', ' Output-Argument ') :- Prolog-Body
   Note: The type definition is a Prolog clause, whose body goal should perform some check on the Input-Argument, create a standardized representation of its value and match it to Output-Argument.

Rule-Procedure ::= 
   Prolog-Functor (' Attribute-Name { ', Attribute-Name } ') 
   :- Prolog-Body
   Note: The different Attribute-Names must denote attributes of the class being defined.

Prolog-Body ::= 
   A Prolog clause body.

Database-Operation ::= 
   Operation Class-Name 'with' Attribute-Value-List

Operation ::= 
   'insert' | 'modify' | 'delete' | 'retrieve'

Attribute-Value-List ::= 
   '[' Attribute-Value-Entry { ',' Attribute-Value-Entry } ']'

Attribute-Value-Entry ::= 
   Attribute-Value-Name ':=' Value { '&', Value }

Value ::= 
   Constant | Variable
   Note: Constant and Variable are Prolog constants and variables.
BIBLIOGRAPHY


CURRICULUM VITAE

I was born in Basel, Switzerland on October 8, 1961. After four years of primary school, I attended the Gymnasium of Mathematics and Natural Sciences and obtained the Matura Certificate in 1979. I studied electrical engineering at the Swiss Federal Institute of Technology (ETH) where I graduated in 1984. In 1983, I completed a six-week full-time course in engineering pedagogics.

I then worked as a research and teaching assistant at the Automatic Control Laboratory of the ETH. At the same time, I attended postgraduate courses on Systems Theory, Artificial Intelligence and Software Engineering. Since 1986, I also worked on the assembly automation project from which this thesis resulted.