DISS. ETH NO. 21863

The Cloud, Paper Planes, and the Cube

A dissertation submitted to

ETH ZURICH

for the degree of

Doctor of Sciences

presented by

GAJAMOHAN MOHANARAJAH

M.Eng. in Mech. and Env. Informatics, Tokyo Institute of Technology
B.Eng. in Control and Systems, Tokyo Institute of Technology

born June 29, 1980
citizen of Sri Lanka

accepted on the recommendation of

Prof. Dr. Raffaello D'Andrea, ETH Zurich, examiner
Prof. Dr. Andreas Krause, ETH Zurich, co-examiner
Prof. Dr. Jonas Buchli, ETH Zurich, co-examiner
Prof. Dr. Bradley Nelson, ETH Zurich, co-examiner

2014

The Cloud, Paper Planes, and the Cube

Gajamohan Mohanarajah

Institute for Dynamic Systems and Control
ETH Zurich
Zurich, March 2014

Cover illustration by Gajamohan Mohanarajah

Institute for Dynamic Systems and Control
ETH Zurich
Switzerland

Printed in Switzerland.

Abstract

This dissertation covers two independent topics. The first topic, cloud robotics, is split into two subtopics, respectively titled *The Cloud*, and *Paper Planes*. *The Cloud* covers the development of a novel cloud infrastructure for robotics, and *Paper Planes* builds on this infrastructure by presenting algorithms that recognize patterns and learn from the data accumulated by robots that are connected to the cloud. The second independent topic, titled *The Cube*, features the development of a small cube-shaped device that can jump up and balance on its edge or corner.

Cloud Robotics

The Cloud: This section describes a robotics-specific Platform-as-a-Service (PaaS) framework, called Rapyuta, that helps robots offload heavy computation by providing secured, customizable computing environments in the cloud. These computing environments allow robots to easily access knowledge repositories that contain data accumulated from other robots; the tight interconnection of these environments thus paves the way for the deployment of robotic teams. A concrete demonstrator - wherein multiple low-cost robots use Rapyuta to perform collaborative 3D mapping in real-time - showcase the framework's key communication and computational features.
Paper Planes: This section features several learning algorithms that were developed to recognize patterns and learn from data accumulated by robots that are connected to a cloud-based framework like Rapyuta. This dissertation focuses specifically on the method of Gaussian process optimization-based learning for trajectory tracking. This work proposes a reinforcement learning algorithm, proves its convergence for trajectory tracking, and shows that a significant amount of knowledge can be transferred even in cases where the reference trajectories are not the same. Simulated flying vehicles are used to numerically illustrate the theoretic results of this work, hence the reference to paper planes in this section's title.

Development of a self-erecting 3D inverted pendulum

The Cube: This section describes the development of a 15 cm sided cube shaped device called the Cubli (Swiss-German for 'small cube'), that can jump up and balance on its edge or corner. Reaction wheels mounted on three faces of the Cubli rotate at high angular velocities and then brake suddenly, allowing the Cubli to jump up, first to its edge, and then to its corner. Once the Cubli has almost reached the balancing position, controlled motor torques are applied to make it balance.

Zusammenfassung

Diese Dissertation behandelt zwei voneinander unabhängige Themen mit den Titeln Cloud Robotics und *The Cube*. Das erste Thema ist in zwei Unterbereiche mit den Titeln *The Cloud* und *Paper Planes* unterteilt. The Cloud behandelt die Entwicklung einer neuen Cloud Infrastruktur für Roboter während Paper Planes auf diese Infrastruktur aufbaut und Algorithmen beschreibt, die es ermöglichen Muster zu erkennen und von Daten zu lernen, die Roboter, welche mit der Cloud verbunden sind, gesammelt haben. Das zweite Thema beschreibt die Entwicklung eines würfelartigen Objekts, das sich selbst aufrichten kann und auf einer seiner Kanten oder Ecken balanciert.

Cloud Robotics

The Cloud: Dieser Abschnitt beschreibt ein Platform-as-a-Service (PaaS) Framework mit dem Namen Rapyuta, welches auf Roboter zugeschnitten ist und diese dabei unterstützt aufwendige Rechenaufgaben auszulagern indem es gesicherte und anpassbare Rechenumgebungen in der Cloud zur Verfügung stellt. Die verschiedenen Rechenumgebungen erlauben es Robotern einfach auf Datenbanken, die das gesammelte Wissen anderer Roboter enthalten, zuzugreifen; die enge Kopplung dieser Umgebungen ermöglicht demnach den Einsatz von Roboter Teams. Ein konkretes Beispiel in dem mehrere kostengünstige Roboter Rapyuta nutzen um im Verbund eine 3D Karte in Echtzeit zu zeichnen demonstriert die wesentlichen Kommunikations- und Rechenmerkmale des Frameworks.

Paper Planes: Dieser Abschnitt behandelt verschiedene Algorithmen, die entwickelt wurden um Muster zu erkennen und aus Daten zu lernen, die von Robotern, welche über ein Cloud-basiertes Framework wie Rapyuta verbunden sind, gesammelt wurden. Diese Dissertation konzentriert sich dabei speziell auf Lernmethoden für die Trajektorienverfolgung, die auf der Gauss'schen Prozess Optimierung basieren. Diese Arbeit präsentiert einen Algorithmus für bestärkendes Lernen, beweist seine Konvergenz bei der Trajektorienverfolgung und zeigt dass ein signifikanter Teil des Wissens sogar dann transferiert werden kann wenn die Referenztrajektorien nicht gleich sind. Simulierte fliegende Vehikel werden benutzt um die theoretischen Resultate dieser Arbeit darzulegen, was die Referenz im Titel dieses Abschnitts erklärt.

Entwicklung eines sich selbst aufrichtenden umgekehrten 3D Pendel

The Cube: Dieser Abschnitt beschreibt die Entwicklung eines würfelartigen Objekts mit dem Namen Cubli (angelehnt an den schweizerdeutschen Diminutiv um einen kleinen Cube zu bezeichnen), das sich selbst aufrichten kann und auf einer seiner Kanten oder Ecken balanciert. Dafür sind an drei Seiten des Cubli Schwungräder befestigt, die mit hoher Geschwindigkeit rotieren und schlagartig bremsen so dass der Cubli sich selbst zunächst auf eine seiner Kanten und dann auf eine seiner Ecken aufrichtet. Sobald sich der Cubli in der Nähe seiner aufrechten Position befindet werden die Motordrehmomente so geregelt dass der Cubli balanciert.

Acknowledgements

This work would not have been possible without the support and contribution from a number of individuals, and here I extend to them my sincerest gratitude.

I owe my deepest gratitude to my advisor Prof. Raffaello D'Andrea for giving me the opportunity to work in his group, introducing me to creative and challenging projects, giving me so much freedom to explore my own ideas, and for helping me to recalibrate my standards on details, mathematical rigor, and simplicity. I feel honoured to be one of Raff's students and I cherish all the interactions I had with him. My gratitude extends to my co-advisor, Prof. Andreas Krause, for taking the time to patiently introduce me to new concepts and ideas in machine learning, as I had a great passion for the field but did not have a strong background in it. Along these lines, I would also like to thank my examiners Prof. Bradley Nelson and Prof. Jonas Buchli for their valuable comments on my thesis.

Next, I would like to thank the RoboEarth team at ETH Zurich, Nico Hubel and Markus Waibel, for the support and memorable times. Nico was the best office mate one could have. Thank you Nico for all the help that extended beyond the research work, and thank you for the wonderful times. My family is going to miss you a lot. Markus was my unofficial advisor. We had hours and hours of exciting discussions on Cloud Robotics and related topics, and we co-organized workshops and wrote many papers together. Thank you, Markus, for your great enthusiasm and support.

I would like to thank all my colleagues at the Institute of Dynamic Systems and Control (IDSC) for the wonderful time and support. Raymond (Ray) Oung and I had some really good times: jogging every week, discussing each other's research, and sharing our personal stories. Thank you, Ray, for being a great trainer and a true friend; you are one of the best things that happened to me in the last four years. Sebastian Trimpe was my role model and a good friend from the beginning, when I first began my internship at IDSC. We were the 'cube' guys, and I am grateful for the many long discussions on balancing cubes, and for your wonderful tilt estimation algorithm. Philipp Reist hosted me at his place when I first arrived in Zurich, was a very helpful friend; Philip, your happy aura and warm hugs always helped me a lot, especially when I was feeling down. Thank you Max Kriegleder, for the fun times at J44 and for the interesting discussions on embedded

systems. Thank you Igor Thommen, Marc-Andre Corzillius, and Hans Ulrich Honegger for all the help with the Cubli.; it would have been impossible without you three. Thank you, Carolina, for your help in making such stand-out graphics and video for the Cubli. I also extend my thanks to the other members of IDSC for for being an inspiration and for the good times, including Angela Schoellig, Sergei Lupashin, Mark Mueller, Markus Hehn, Luca Gherardi, Dario Brescianini, Robin Ritz, and Michael Hammer, my awesome new office-mate after PhD. A special thanks goes to our institute secretary Katharina Munz for all the help, care, and the fun times we had outside the lab training for the SOLA race. I would also like to say a big thanks to Hallie Siegel for patiently going through almost all my writings (including this acknowledgement) and giving valuable feedback. And finally, I would like to thank Raff once again for giving me the opportunity to spend time with such a great team for four years.

One of my greatest joys during my PhD was collaborating with students. Dominique Hunziker and Okan Koc somehow managed to spend two years with me. I really enjoyed the time with these two and I learned a lot from both. Thank you, Vlad, for giving a boost to Rapyuta with the mapping demonstrator. Thank you, Michael Merz, for giving Cubli a great start, Tobias Widmer for making it balancing, and Christof Dubs for a great finish with 4 million views. Thank you, Michael Mulebach, for the strong theoretical contribution to the Cubli project and the in-depth discussions. I will never forget our emergency meeting on non-zero dynamics for the CDC deadline at the university hospital when Kalai was trying to give birth.

I would also like to express my gratitude to ETH Zurich and the European Commission for their support and funding of my research.

In addition to the above, there were many people who educated me, motivated me, and helped me along the way.

Among those, I would like to thank the people of Japan for giving me a full scholarship to study in their country and making my robotics dream come true. Specifically, I would like to thank my Japanese language teachers Kusakari-sensei and Tachizono-sensei and many others who, in just one year, taught me to understand my university lectures. Next, I would like to thank all the teachers of Kurume National College of Technology. Thank you, Esaki-sensei, Kuroki-sensei, Kumamaru-sensei, Fukuda-sensei, Kawaguchi-sensei, Ayabe-sensei, Sakuragi-sensei, Amafuji-sensei, and Noda-san for teaching me strong basics and for the huge mental support in the early years. Thank you Teoh and Daniel for the great company. I would also like to thank all my teachers at Tokyo Institute of Technology. I especially would like to thank Hayakawa-sensei, my bachelor and master thesis supervisor, for the time he patiently spent to introduce me to the world of research and scientific writing. I feel very privileged to be one of his first students and had a very memorable and educational time in his laboratory. I would also like to thank my friends in Japan, especially Sriram Iyer, for being a great senpai (senior), friend, and a role model for me to go to Japan. I would also like the thank Arul, for being an awesome flat-mate and my AOL buddies, Nikhil, Keeru, Amit, Mohan-san,Meena-san, Ani, Ajith, Shubra, Naveen, Sangeetha-san, Poornima-san, Shreya, Kyoko-san, Neela-san, and my Japanese okasan

Kayo-san. Thanks to you guys, Japan was like my second home.

A big thanks also goes to all my teachers in Sri Lanka. Starting with Ms. Sumathy Ranjakumar, for being a lovely and supportive class teacher, Mr. Gnanasundaram and Mr. Premnath my math gurus, and Mr. Soundararajan my physics teacher. My deepest gratitude goes to Ms. Velupillai, my chemistry teacher, who voluntarily gave me intense private lessons free of charge after noticing my poor performance in the trial exams. This was very pivotal step in my career and I would like to dedicate this thesis to her.

I would like thank my family for always being with me. Amma and Appa, thank you for your selfless effort to give Latha and me the best education, and for your unconditional love. Latha, thank you being an inspiration to me. The biggest contribution to this thesis is comes from my better half, Kalai. Thank you, Kalai, for your continuous care, motivation, and love. Without you, I would not have enjoyed life in Zurich and this thesis would not have been possible in its present form. Finally, Atchuthan (I know you still can not read) for being the big bundle of joy that kept me going during the last year of my PhD.

Contents

1

Introduction

This dissertation describes the work done in two distinct topic areas: 1) cloud robotics and associated learning algorithms, as described in *The Cloud* and in *Paper Planes*[1] subsections, and 2) development of a self-erecting 3D inverted pendulum, as described in *The Cube* subsection.

Cloud Robotics

The past decade has seen the first successful, large-scale use of mobile robots. However, the vast majority of these robots continue to either use simple control strategies (e.g., robot vacuum cleaners) or be operated remotely by humans (e.g., drones, unmanned ground vehicles, telepresence robots). One reason these mobile robots lack intelligence is because the costs of onboard computation and storage are high; this affects not only the robot's price point, but also results in the need for additional space and extra weight, which constrain the robot's mobility and operation time. Another reason is the absence of a common mechanism and medium to communicate and share knowledge between robots with potentially different hardware and software components.

Cloud robotics is an emerging sub-discipline that aims to solve some of the aforementioned challenges. It is a field rooted in cloud computing, cloud storage, and other internet technologies that are centered around the benefits of converged infrastructure and shared resources. It allows the robots to benefit from the powerful computational, storage, and communication resources of modern data centers. In addition, it removes overheads for maintenance and updates, and reduces dependence on custom middleware.

RoboEarth [1], a pioneering cloud robotics initiative, focuses on the following three topics and related questions in order to build an internet for robots:

- *Knowledge representation*: What types of knowledge should be shared between robots? How can knowledge be represented in a platform-independent manner?

[1] One of the learning methods used simulated flying vehicles to numerically illustrate the theoretical results, thus the name paper planes.

How can this knowledge be used for reasoning? How can common knowledge be used across heterogeneous platforms?

- *Storage*: What is the best infrastructure to store the semantic and binary forms of data/knowledge? How can new knowledge be created from data gathered from different robots? How can knowledge be transferred between two robots?
- *Computation*: How can a scalable cloud-based architecture that allows robots to offload some of their computation to the cloud be built? What are the tradeoffs between onboard and cloud-based execution?

This dissertation contributed to the general knowledge of cloud robotics by 1) developing a novel cloud robotics platform, as described below in *The Cloud*, and 2) developing new algorithms that are able to learn from a robots' accumulated experience and transfer the learnt knowledge between robots, as described below in *Paper Planes*.

The Cloud

Cloud Robotics allows robots to take advantage of the rapid increase in data transfer rates to offload computationally expensive tasks, see Fig. 1.1. This is of particular interest for mobile robots where on-board computation entails additional power requirements that may reduce operating time, constrain robot mobility, and increase costs.

Running robotics applications in the cloud falls into the Platform-as-a-Service (PaaS) model [2] of the cloud computing literature. In PaaS the cloud computing platform typically includes an operating system, an execution environment, a database, and a communication server. Many existing cloud computing building blocks - including much of the existing hardware and software infrastructure for computation, storage, network access, and load balancing - can be directly leveraged for robotics. However, specific requirements (such as the need for multi-process applications, asynchronous communication, and compatibility with existing robotics application frameworks) limit the applicability of existing cloud computing platforms to robot application scenarios.

The idea of having a remote brain for the robots can be traced back to the 90s [3, 4]. During the past few years, this idea has gained traction (mainly due the availability of computational/cloud infrastructures), and several efforts to build a cloud computing framework for robotics have emerged [5]–[7]. The open source project Rapyuta[2] attempts to solve some of the remaining challenges of building a complete cloud robotics platform.

Rapyuta allows to outsource some or all of a robot's onboard computational processes to a commercial data center. It is distinguished from other similar frameworks (like the Google App Engine) in that it is specifically tailored to multi-process, high-bandwidth robotics applications and middleware, and provides a well-documented open source implementation that can be modified to cover a large variety of robotic scenarios. Rapyuta supports out-of-the-box outsourcing of almost all the current 3000+ ROS

[2]The name is inspired from the movie *Tenku no Shiro Rapyuta* (English title: Castle in the Sky) by Hayao Miyazaki, where Rapyuta is the castle in the sky inhabited by robots.

packages and is easily extensible to other robotic middleware. A pre-installed Amazon Machine Image (AMI) allows Rapyuta to be launched in any of Amazon's data centers within minutes. Once launched, robots can authenticate themselves to Rapyuta, create one or more secured computational environments in the cloud, and launch the desired nodes/processes. The computational environments can also be arbitrarily connected to build parallel computational architectures on the fly. The WebSocket-based communication protocol, which provides synchronous and asynchronous communication mechanisms, allows not only ROS-based robots to connect to the ecosystem, but also browsers and mobile phones to connect as well. Target applications include collaborative 3D mapping, task/grasp planning, object recognition, localization, and teleoperation, among others. Rapyuta provides secure, private computing environments and optimized data throughput. However, its performance is in large part determined by the latency and quality of the network connection and the performance of the data center. Optimizing performance under these constraints is typically highly application-specific. In Chapter 3, this dissertation demonstrates performance optimization for a collaborative 3D real-time mapping scenario.

Paper Planes

Within a cloud-based computation and storage framework, learning algorithms play a crucial part in accumulating all the data, recognizing patterns, and producing knowledge. The ability to transfer knowledge between different robots and contexts (environments/problems) will significantly improve the performance of future robots connected to a RoboEarth-like system.

For example, consider a robot learning how to pour tea into a cup and over time perfecting its motions. The learned pouring motion can be uploaded to a central database, annotated with the hardware-specifics of the particular robot as well as the size and shape of the teapot as context. Another robot with slightly different hardware, holding a different teapot, can download the stored motion as a prior and adapt it to its particular context, thereby eliminating the need to learn the motion from scratch.

Systems that work in a repetitive manner, such as robotic manipulators and chemical plants, use Iterative Learning Control (ILC) to iteratively improve the performance over a given repeated task or trajectory. The feed-forward control signal is modified in each iteration to reduce the error or the deviation from the given reference trajectory. A good analogy is a basketball player shooting a free throw from a fixed position: during each shot the basketball player can observe the trajectory of the ball and alter the shooting motion in the next attempt [12]. A key limitation with ILC is that it assumes the task or the trajectory to be fixed (constant) over iterations. While this is a reasonable assumption for some repeated tasks, ILC cannot handle the cases when the trajectory is modified or changing over time, and the controller must start learning from scratch. It is shown in Chapter 4 that a significant amount of knowledge can be transferred even between cases where the reference trajectories are not the same. Basketball players do not have to learn the free throw motion from scratch each time they find themselves in a slightly different

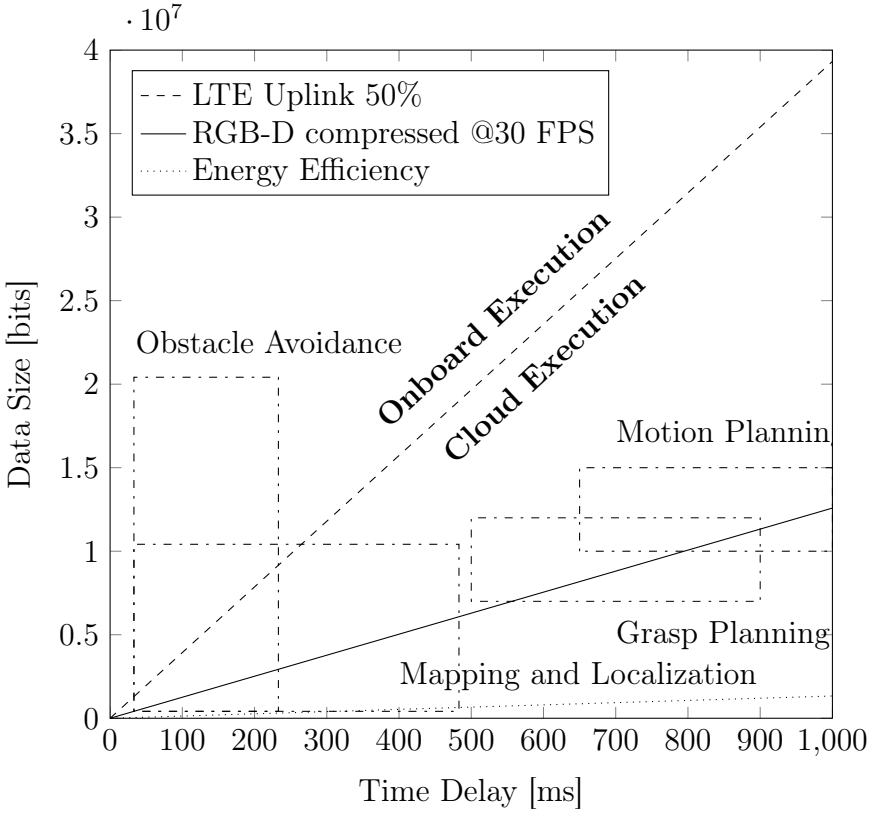


Figure 1.1: Consumed data size against the delay deadline of various robotics applications [8]–[10]. The data transfer rate (dashed line) is an important factor for deciding if an application is feasible for deployment in the cloud. Increased data rates (depicted by an increased slope in the dashed line) indicate improved feasibility of the cloud application. In terms of energy consumption, cloud execution is favourable to onboard execution for any task or application lying above the dotted line. For details on energy consumption see [11].

position. We call these cases or reference trajectories *contexts*. Context can change in a given task, and it is the responsibility of the autonomous agent or the learning controller to adapt to different contexts.

In Chapter 4, this dissertation introduces a reinforcement-learning (RL) algorithm that learns to track trajectories in state space online. Specifically, the proposed algorithm uses Gaussian Process optimization in the bandit setting to track a given trajectory. It implicitly learns the dynamics of the system, and in addition to improving the tracking performance, it facilitates knowledge transfer between different trajectories.

Development of a Self-erecting 3D Inverted Pendulum

The Cube

The Cubli is a $15 \times 15 \times 15$ cm cube that can jump up and balance on its corner. Reaction wheels mounted on three faces of the cube rotate at high angular velocities and then brake suddenly, causing the Cubli to jump up. Once the Cubli has almost reached the corner stand up position, controlled motor torques are applied to make it balance on its edge or corner. In addition to balancing, the motor torques can also be used to achieve a controlled fall, such that the Cubli can be commanded to fall in any arbitrary direction. Combining these three abilities – jumping up, balancing, and controlled falling – the Cubli is able to move across a surface using only internal actuation. To date, Cubli is the smallest 3D inverted pendulum, and the first 3D inverted pendulum that can self erect. See Chapter 5 for more information on the Cubli's mechatronic and algorithmic design.

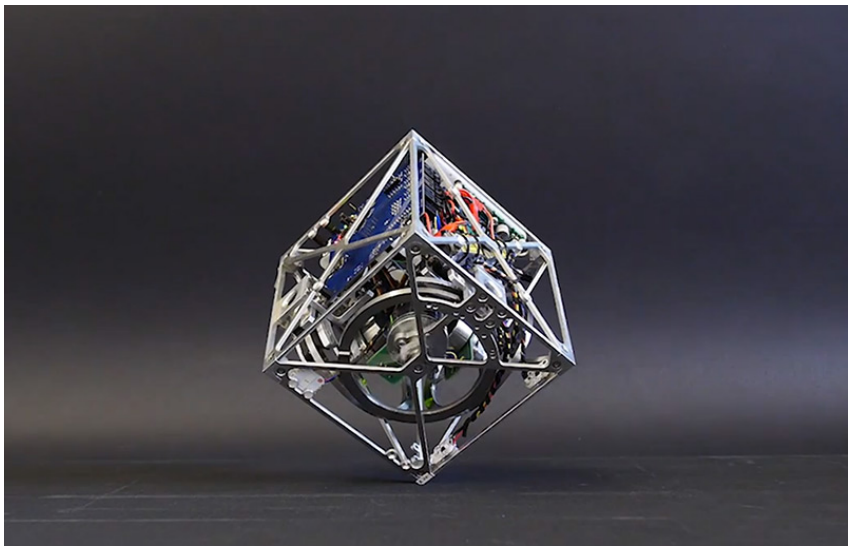


Figure 1.2: The Cubli balancing on its corner.

References

[1] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft, "RoboEarth," *Robotics Automation Mag., IEEE*, vol. 18, no. 2, pp. 69 –82, June 2011.

[2] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Special Publication 800-145, 2011, available http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[3] K. Goldberg and R. Siegwart, Eds., *Beyond webcams: an introduction to online robots*. Cambridge, MA, USA: MIT Press, 2002.

[4] M. Inaba, S. Kagami, F. Kanehiro, Y. Hoshino, and H. Inoue, "A platform for robotics research based on the remote-brained robot approach." *I. J. Robotic Res.*, vol. 19, no. 10, pp. 933–954, 2000.

[5] R. Arumugam, V. R. Enti, K. Baskaran, and A. S. Kumar, "DAvinCi: A cloud computing framework for service robots," in *Proc. IEEE Int. Conf. Robotics and Automation*. IEEE, May 2010, pp. 3084–3089.

[6] K. Kamei, S. Nishio, N. Hagita, and M. Sato, "Cloud Networked Robotics," *Network, IEEE*, vol. 26, no. 3, pp. 28–34, May-June 2012.

[7] M. Sato, K. Kamei, S. Nishio, and N. Hagita, "The ubiquitous network robot platform: Common platform for continuous daily robotic services," in *System Integration (SII), 2011 IEEE/SICE Int. Symp.*, Dec 2011, pp. 318 –323.

[8] B. Kehoe, D. Berenson, and K. Goldberg, "Toward cloud-based grasping with uncertainty in shape: Estimating lower bounds on achieving force closure with zero-slip push grasps," pp. 576–583, May 2012.

[9] M. Tenorth, U. Klank, D. Pangercic, and M. Beetz, "Web-enabled robots," *Robotics & Automation Magazine, IEEE*, vol. 18, no. 2, pp. 58–68, 2011.

[10] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, "Learning, planning, and control for quadruped locomotion over challenging terrain," no. 2, pp. 236–258, 2010.

[11] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *Network, IEEE*, vol. 26, no. 3, pp. 21–28, May-June 2012.

[12] D. Bristow, M. Tharayil, and A. Alleyne, "A survey of iterative learning control," *Control Systems, IEEE*, vol. 26, no. 3, pp. 96 – 114, June 2006.

Contribution and Organization

The bulk of this dissertation is comprised of four main chapters, Chapters 2-5, which can be read independently. The chapters draw their information mainly from the author's publications and technical reports. The specific contributions made in each of the subsequent chapters, as well as their references, are listed below.

Chapter 2: Rapyuta
This chapter details the design and implementation of Rapyuta, an open source Platform-as-a-Service (PaaS) framework designed specifically for cloud robotics applications. Rapyuta helps robots to offload heavy computation by providing secured customizable computing environments in the cloud. The computing environments also allow the robots to easily access the RoboEarth knowledge repository. Furthermore, these computing environments are tightly interconnected, paving the way for deployment of robotic teams. The chapter also describes three typical use cases, some benchmarking and performance results, and two proof-of-concept demonstrators.

Major contributions described in this chapter include:

- Design and implementation of an open-source cloud robotics platform
- Benchmarking of the system's functionalities with other platforms
- Two proof-of-concept demonstrators

The framework developed under this work was extensively used in the RoboEarth project by other partner institutions. During the final demonstration of the project, the framework was used to run the following components:

- WIRE, world modelling component, Eindhoven University of Technology,
- C^2TAM, cloud-based mapping components, University of Zaragoza,
- Multi-robot planning component, University of Stuttgart,
- Human Machine Interface Components, Technical University of Munich, and
- KnowRob Reasoning Engine, University of Bremen.

In addition, the framework is used by 10+ researchers/corporate research groups/start-ups around the world. The work in this chapter was disseminated via several talks and workshops:

- **Invited Talk: ICRA 2013 Workshop on Long-Term Autonomy**, "The RoboEarth Cloud: Powering Long-term Autonomy," Karlsruhe, Germany, May 2013, Gajamohan Mohanarajah (ETH Zurich)
- **Invited Talk: Oracle Partner Network Lounge**, "Cloud Robotics," Geneva/Zurich, Switzerland, June 2013, Gajamohan Mohanarajah/Dominique Hunziker (ETH Zurich)

- **ROSCon 2013**, "Understanding the RoboEarth Cloud," Stuttgart, Germany, May 2013, Gajamohan Mohanarajah (ETH Zurich)
- **euRobotics Forum 2013, Cloud Robotics Workshop**, Gajamohan Mohanarajah (ETH Zurich), Oliver Zweigle (University of Stuttgart), Alexander Perzylo (Technical University of Munich), Markus Waibel (ETH Zurich), (Lyon, France)
- **IROS 2013, Cloud Robotics Workshop**, Organizers: Markus Waibel, (ETH Zurich-main organizer), Ken Goldberg (UC Berkeley), Javier Civera (Uni. Zaragoza), Alper Aydemir (NASA JPL), Matei Ciocarlie (Willow Garage), Gajamohan Mohanarajah (ETH Zurich) (Tokyo, Japan)

Finally, this work was awarded the **Amazon Web Services in education grant award** for its pioneering role in cloud robotics. The results contained in this chapter were published in

[1] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel, "Rapyuta: A cloud robotics platform," *IEEE Transactions on Automation Science and Engineering **(accepted)***, February 2014.

[2] D. Hunziker, G. Mohanarajah, M. Waibel, and R. D'Andrea, "Rapyuta: The RoboEarth Cloud Engine," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013, pp. 438–444.

Chapter 3: Cloud-based collaborative mapping in real-time with low-cost robots

This chapter describes a concrete robotics application that was built to demonstrate various functionalities of the Rapyuta framework described in Chapter 2. More specifically, this chapter describes an architecture, protocol, and parallel algorithms for collaborative 3D mapping in the cloud. The robots run a dense visual odometry algorithm on a smartphone-class processor. Key-frames from the visual odometry are sent to the cloud for parallel optimization and merging with maps produced by other robots. After optimization the cloud pushes the updated poses of the local key-frames back to the robots. All processes are managed by Rapyuta, a cloud robotics framework, running in a commercial data center. Finally, this chapter presents qualitative visualization of collaboratively built maps, as well as quantitative evaluation of localization accuracy, bandwidth usage, processing speeds, and map storage.

Major contributions described in this chapter include:

- Open source parallel implementation of dense visual odometry on a smartphone-class ARM multi-core CPU
- A cloud-based SLAM architecture and protocol that significantly reduces the bandwidth usage

- Techniques for parallel map optimization and merging over multiple machines in a commercial data center
- An experimental demonstrator for quantitative and qualitative evaluation of the proposed methods

The results contained in this chapter will be submitted to:

[1] G. Mohanarajah, V. Usenko, M. Singh, M. Waibel, and R. D'Andrea, "Cloud-based collaborative 3D mapping in real-time with low-cost robots," *IEEE Transactions on Automation Science and Engineering* ***(accepted)***, March 2014.

Chapter 4: Learning Methods
The following two learning methods were motivated by and developed for RoboEarth. The first of these methods is described in detail in this chapter.

- **Gaussian Process Optimization-based Learning for Trajectory Tracking**: Systems that work in a repetitive manner use Iterative Learning Control (ILC) algorithms to iteratively improve the performance of a given task or trajectory over time. The limitation with ILC is that it assumes the task or the trajectory to be fixed over iterations. ILC cannot handle cases when the trajectory is modified or changing over time, and the iterative learning controller must start learning from scratch. This work presents a reinforcement learning algorithm, proves its convergence for trajectory tracking, and shows that a significant amount of knowledge can even be transferred between cases where the reference trajectories are not the same.
- **Indirect Object Search**: The Indirect Object Search algorithm developed in this work provides two models for predicting the occurrence and location probabilities of small and hard-to-detect object classes based on the occurrence and location of large, easy-to-detect object classes. All relationship models were trained with a large dataset, which consisted of 1449 well-annotated images of indoor scenes captured with a Microsoft Kinect. The software implementation of this algorithm was used in RoboEarth's final demonstrator to assist the service robots by giving a prior on the likely locations of small hard-to-detect objects. This method is not covered in this dissertation. For details on this method see [2].

Other algorithmic work done under this topic include, articulation model learning and Iterative Learning Control. In articulation model learning (2011) the robots learnt the articulation models of various furniture items and shared the learnt models with other robots using RoboEarth. Iterative Learning Control was used for trajectory tracking in mobile soccer robots during the first demonstrator (2010) of the RoboEarth project.
Chapter 5: The Cubli
This chapter focuses on the Cubli and describes the mechatronic design, state estimation algorithm, system identification procedure, nonlinear control design, learning strategy

for jump up, and finally, the experimental results of the system. Major contributions described in this chapter include:

- State-of-the-art mechatronic design of what is to date the smallest 3D inverted pendulum
- A system identification technique that does not require additional apparatus
- A non-linear control design
- The first set of experimental results demonstrating internally actuated motion under earth's gravity

The results contained in this chapter were published and/or will be submitted to:

[1] G. Mohanarajah, C. Dubs, and R. D'Andrea, "The Cubli," *Mechatronics **(in preparation)***, December 2014.

[2] M. Muehlebach, G. Mohanarajah, and R. D'Andrea, "Nonlinear analysis and control of a reaction wheel-based 3D inverted pendulum," in *Proc. IEEE Conference on Decision and Control (CDC)*, Florence, Italy, 2013, pp. 1283–1288.

[3] G. Mohanarajah, M. Muehlebach, T. Widmer, and R. D'Andrea, "The Cubli: A reaction wheel-based 3D inverted pendulum," in *Proc. European Control Conference (ECC)*, Zurich, Switzerland, 2013, pp. 268–274.

[4] G. Mohanarajah, M. Merz, I. Thommen, and R. D'Andrea, "The Cubli: A cube that can jump up and balance," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura-Algarve, Portugal, 2012, pp. 3722–3727.

This dissertation concludes with Chapter 6 by taking a retrospective of the work accomplished. It discusses in a broad sense open problems, both practical and theoretical, and future research directions. Conclusions that summarize the specific contributions and/or provide direct extensions to the work are presented at the end of each chapter.

2

Rapyuta: A Cloud Robotics Framework

2.1 Introduction

The past decade has seen the first successful, large-scale use of mobile robots. However, the vast majority of these robots either continue to use simple control strategies (e.g., robot vacuum cleaners) or are operated remotely by humans (e.g., drones, unmanned ground vehicles, telepresence robots). One reason these mobile robots lack intelligence is because the costs of onboard computation and storage are high; this affects not only the robot's price point, but also results in the need for additional space and extra weight, which constrain the robot's mobility and operation time. Another reason is the absence of a common mechanism and medium to communicate and share knowledge between robots with potentially different hardware and software components.

The rapid progress of wireless technology and availability of data centers hold the potential for robots to tap into the cloud. Using the web as a powerful computational resource, a communication medium, and a source of shared information could allow developers to overcome these current limitations by building powerful cloud robotics applications. Example applications include map building [1], task/grasp planning [2], object recognition, localization, and many others. Cloud robotics applications hold the potential for lighter, smarter and more cost-effective robots.

Running robotics applications in the cloud falls into the Platform-as-a-Service (PaaS) model [4] of the cloud computing literature. In PaaS the cloud computing platform typically includes an operating system, an execution environment, a database, and a communication server. Many existing cloud computing building blocks, including much of the existing hardware and software infrastructure for computation, storage, network access,

This paper is accepted for publication in the *IEEE Transactions on Automation Science and Engineering*, February 2014.

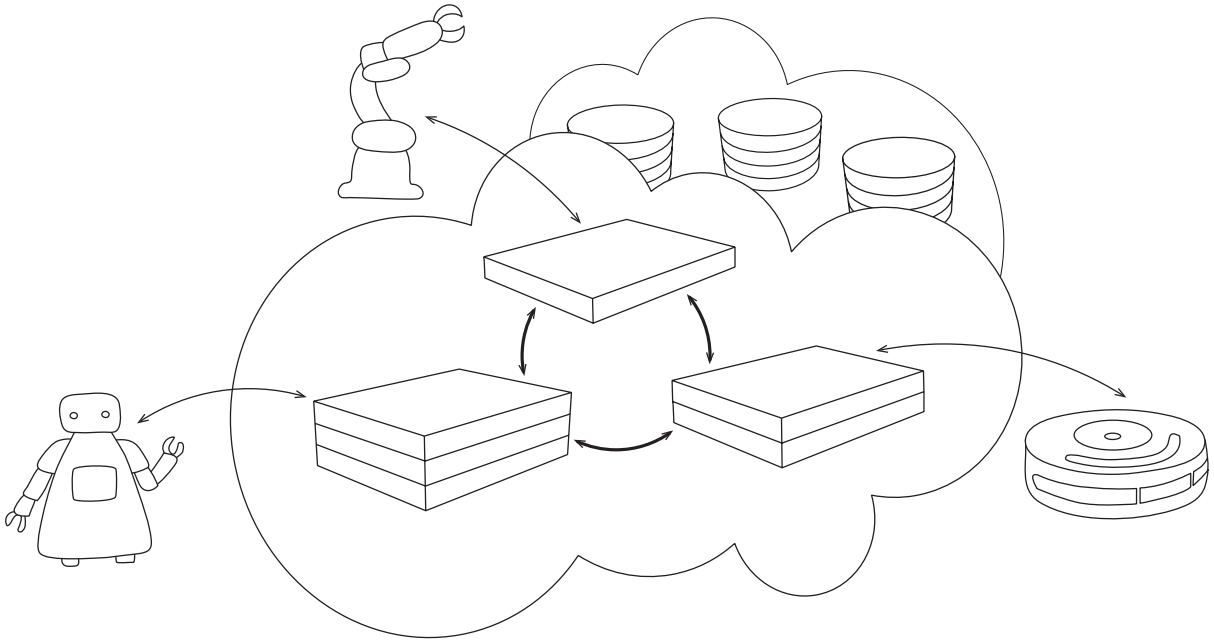


Figure 2.1: Simplified overview of the Rapyuta framework: Each robot connected to Rapyuta has one or more secured computing environments (rectangular boxes) giving them the ability to move their heavy computation into the cloud. In addition, the computing environments are tightly interconnected with each other and have a high bandwidth connection to the RoboEarth [3] knowledge repository (stacked circular disks).

and load balancing, can be directly leveraged for robotics. However, specific requirements (such as the need for multi-process applications, asynchronous communication, and compatibility with existing robotics application frameworks) limit the applicability of existing cloud computing platforms to robot application scenarios. For example, a general PaaS platform such as the popular Google App Engine [5] is not well suited for robotics applications since it exposes only a limited subset of program APIs required for a specific web application, allows only a single process, and does not expose sockets, which are indispensable for robotic middlewares such as ROS [6].

The popular PaaS framework Heroku [7] overcomes some of these limitations, but lacks features required for many robotics applications, such as multi-directional data flow between robots and their computing environments. Other PaaS frameworks such as Cloud Foundry [8] and OpenShift [9] offer more flexibility and may prove useful for some robotics applications in the future. However, fundamental differences in the requirements of human vs. robot users, such as typical uplink rates and speeds, may lead to different trade-offs and design choices, and may ultimately result in different software solutions for cloud computing and cloud robotics platforms.

The idea of having a remote brain for the robots can be traced back to the 90s [10,11]. During the past few years, this idea has gained traction (mainly due the availability

Please see `http://goo.gl/XGjsT` for a detailed discussion on Rapyuta vs. non-specific PaaS.

of computational/cloud infrastructures), and several efforts to build a cloud computing framework for robotics have emerged. The DAvinCi Project [1] used ROS as the messaging framework to get data into a Hadoop cluster, and showed the advantages of cloud computing by parallelizing the FastSLAM algorithm [12]. It used a single computing environment without process separation or security; all inter-process communications were managed by a single ROS master. Unfortunately, the DAvinCi Project is not publicly available. While the main focus of DAvinCi was computation, the ubiquitous network robot platform (UNR-PF) [13, 14] focused on using the cloud as a medium for establishing a network between robots, sensors, and mobile devices. The project also made a significant contribution to the standardization of data-structures and interfaces. Finally, rosbridge [15], an open source project, focused on the external communication between a robot and a single ROS environment in the cloud.

With the open source project Rapyuta we attempt to solve some of the remaining challenges of building a complete cloud robotics platform. Rapyuta is based on an elastic computing model that dynamically allocates secure computing environments (or clones [16]) for robots. These computing environments are tightly interconnected, allowing robots to share all or a subset of their services and information with other robots. This interconnection makes Rapyuta a useful platform for multi-robot deployments such as those described in [17].

Furthermore, Rapyuta's computing environments provide high bandwidth access to the RoboEarth [3] knowledge repository, enabling robots to benefit from the experience of other robots. Note that until now robots directly submitted and queried data in the RoboEarth repository, and all the processing, planning, and reasoning on this data happened locally on the robot. With Rapyuta, robots can perform these tasks in the cloud by having a corresponding software agent/clone. Thus, Rapyuta is also called the RoboEarth Cloud Engine.

Rapyuta's ROS-compatible computing environments allow it to run almost all open source ROS packages (there are currently more than 3000) without any modifications while sidestepping the severe drawbacks of client-side robotics applications, including requirements for expensive and/or power-hungry hardware, configuration/setup overheads, dependence on custom middleware, as well as often failure-prone maintenance and updates. In addition to its out-of-the-box ROS compatibility, Rapyuta can also be customized for other robotics middlewares.

Finally, Rapyuta's WebSocket-based communication server provides bidirectional, full duplex communications with the physical robot. Note that this design choice also allows the server to initiate the communication and send data or commands to the robot.

The remainder of this paper is structured as follows: Taking a bottom-up approach we present each of the main components of the architecture individually along with our design choices in Sec. 2.2 and Rapyuta's communication protocols in Sec. 2.3. Sec. 2.4 returns to a general picture and presents several use cases that combine the previous components

Rapyuta is part of the RoboEarth initiative aimed at building a world wide web for robots. Visit http://www.roboearth.org/ for details.

and the communication protocols in different ways to fit a variety of deployment scenarios. Then, performance and benchmarking results are presented in Sec. 2.5. This is followed by two robotics demonstrators that highlight various aspects of Rapyuta in Sec. 2.6. We conclude in Sec. 4.5 with a with a brief outlook on Rapyuta's future developments and the potential future of cloud robotics in general.

2.2 Main Components

Rapyuta's four main components are: the computing environments onto which robots offload their tasks, a set of communication protocols, four core task sets to administer the system, and a command data structure to organize the system administration.

Computing Environments

Rapyuta's computing environments are implemented using Linux Containers [18], which provide a lightweight and customizable solution for process separation, security, and scaling. In principle, Linux Containers can be thought of as an extended version of `chroot` [19], which isolates processes and system resources within a single host machine. Since Linux Containers do not emulate hardware (similar to platform virtualization technologies), and since all processes share the same kernel provided by the host, applications run at native speed.

Furthermore, Linux Containers also allow easy configuration of disk quotas, memory limits, I/O rate limits, and CPU quotas, which enables a single environment to be scaled up to fit the biggest machine instance of the IaaS [4] provider, or scaled down to simply relay data to the Hadoop [20] backend, similar to the DAvinCI [1] framework.

Each computing environment is set up to run any process that is a ROS node, and all processes within a single environment communicate with each other using the ROS inter-process communication. Having the well-established ROS protocol inside the environments allows them to run all existing ROS packages without any modifications, and lowers the hurdle for application developers.

Communication and Protocols

One of the basic building blocks of Rapyuta's communication architecture is the *Endpoint*, which represents a process that consists of *Ports* and *Interfaces*. Figure 2.2 shows these building blocks and the basic communication channels of Rapyuta.

Interfaces are used for communication between a Rapyuta process and a non-Rapyuta process running either on the robot or in the computing environment. They provide a synchronous (service-based) or an asynchronous (topic-based) transport mechanisms. *Interfaces* used for communication with robots provide converters, which convert a data message from the internal communication format to a desired external communication format and vice versa. *Ports* are used for communication between Rapyuta processes.

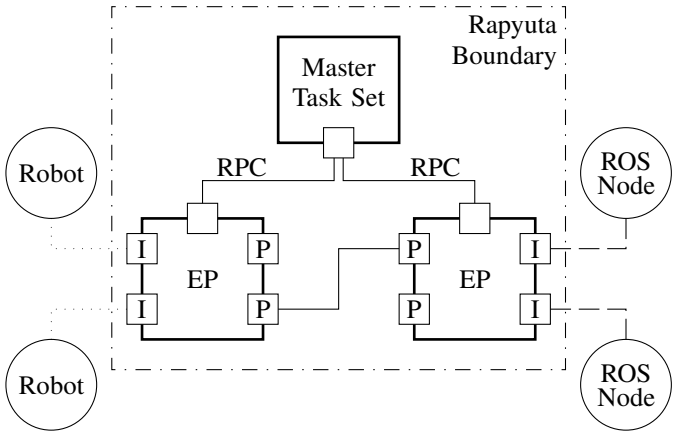


Figure 2.2: The basic communication channels of Rapyuta: The *Endpoints* (EP) are connected to the *Master* task set using a two-way remote procedure call (RPC) protocol. Additionally, the *Endpoints* have *Interfaces* (I) for connections to robots or (ROS) nodes, as well as *Ports* (P) for communication between *Endpoints*. The dotted lines represent the external communication, dashed lines represent the ROS-based communication between ROS nodes and Rapyuta, and finally all solid lines represent the internal communication between Rapyuta's processes.

The *Endpoints* allow the communication protocols to be split into three parts. The first part is the internal communication protocol, which covers all communication between Rapytua's processes. The next part is the external communication protocol, which covers the data transfer between the physical robot and the cloud infrastructure running Rapyuta. The last part consists of the communication between Rapyuta and the applications running inside the containers. Each of these protocols are presented in more detail in Sec. 2.3.

Core Task Sets

This sub-section presents the four Rapyuta task sets that administer the system. A task set is a set of functionalities and one or more of these sets can be put together to run as a process depending on the use case (see Sec. 2.4)

Master Task Set The *Master* task set is the main controller that monitors and maintains the *command* data structure, which includes:

- organization of connections between robots and Rapyuta,
- processing of all configuration requests from robots, and
- monitoring the network of other task sets.

As opposed to the other task sets, only a single copy of the *Master* task set runs inside Rapyuta.

Robot Task Set The robot task set is defined by the capabilities necessary to communicate with a robot. It includes:

- forwarding of configuration requests to the *Master*,
- conversion of data messages, and
- communication with robots and other *Endpoints*.

Environment Task Set The environment task set is defined by the capabilities necessary to communicate with a computing environment. It includes:

- communication with ROS nodes and other *Endpoints*,
- launching/stopping ROS nodes, and
- adding/removing parameters.

A process containing the environment task set runs inside every computing environment.

Container Task Set The container task set is defined by the capabilities necessary to start/stop computing environments. A process containing the container task set runs inside every machine.

Command Data Structure

Rapyuta is organized in a centralized command data structure. This data structure is managed by the *Master* task set and it consists of the four components shown in Fig. 2.3.

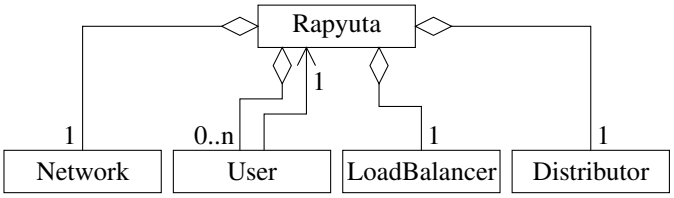


Figure 2.3: Simplified UML diagram of Rapyuta's top level command data structure.

The *Network* (see Fig. 2.4) is the most complex part of the data structure. Its elements are used to provide the basic abstraction of the whole platform and are referenced by the *User*, *LoadBalancer*, and *Distributor* components. The *Network* is also used to organize the internal and external communication, which will be discussed in detail in Sec. 2.3. The addition of *Namespaces* in the command data structure enables an *Endpoint* to group *Interfaces* of a single robot or a computing environment and the addition of the connection classes (*EndpointConnection*, *InterfaceConnection*, and *Connection*) simplifies the reference counting for the connections.

The *User* (see Fig. 2.5) generally represents a human who has one or more robots that need to be connected to the cloud. Each *User* has a unique API key, which is used

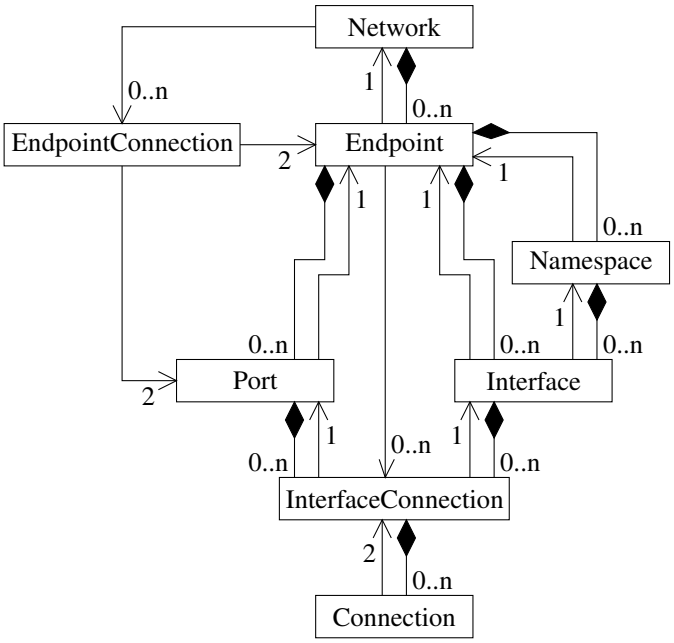


Figure 2.4: Simplified UML diagram of Rapyuta's top level component *Network*.

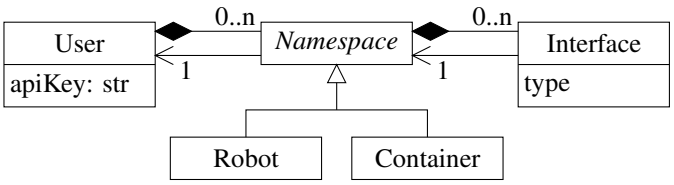


Figure 2.5: Simplified UML diagram of Rapyuta's top level component *User*.

by the robots for authentication. The *User* can have multiple *Namespaces* which, in turn, can have several *Interfaces*.

The *LoadBalancer* (see Fig. 2.6) is used to manage the *Machines* which are intended to run the computing environments. To allow these computing environments to communicate directly with each other without Rapyuta (see Sec. 2.3) the computing environments can be added to a *NetworkGroup*. Therefore the *NetworkGroups* have a representation of each *Container* included in the group and references to the participating *Machines*. Similarly, the *Machines* have a reference of each *Container* they are running. Additionally, the *LoadBalancer* is used to assign new containers to the appropriate machine.

Finally, the *Distributor* is used to distribute incoming connections from robots between available robot *Endpoints*.

2.3 Communication Protocols

This section presents Rapyuta's internal and external communication protocols in more detail.

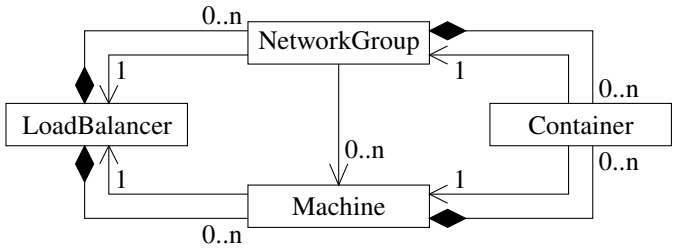


Figure 2.6: Simplified UML diagram of Rapyuta's top level component *LoadBalancer*.

Internal Communication Protocol

All Rapyuta processes communicate with each other over UNIX sockets and the protocol is built using the Twisted framework [21], an event-driven networking engine that uses asynchronous messaging. The type of messages used for the internal communication can be split into two categories. The first type consists of all administrative messages used to configure Rapyuta. All these messages either originate or end in the *Master* process (runs the *Master* task set) containing the command data structure. The Perspective Broker, a two-way RPC implementation for the Twisted framework, is used as the protocol for administrative messages. The second and the most frequent type is the data protocol. For this type of communication, a length prefixed protocol is used. The content of a data message is a serialized ROS message. For Rapyuta, an additional header containing the ID of the sending *Interface*, an optional destination ID (necessary for service type interfaces), and the message ID (which is used also for the external communication) is added. This results in a header length of 22 or 38 bytes plus the message ID, which has a length upper bounded by 255 bytes.

External Communication Protocol

The robots connect to Rapyuta using the WebSockets protocol [22], similar to rosbridge [15]. The protocol was implemented using the Autobahn tools [23], which is also based on the Twisted framework [21]. Unlike a common web server, which uses pull technology, the use of WebSockets allows Rapyuta to push results. Note that this protocol is very general compared to the ROS protocol used in the DAvinCI [1] framework, allowing easy integration of non-ROS robots, mobile devices and even web browsers into the system.

The messages between the robot and Rapyuta are pure ASCII JSON messages that have the following top level structure:

```
{ "type":"...", "data": ... },
```

RPC (Remote Procedure Call) is a communication protocol that allows a process to execute a procedure in another process.

JSON (JavaScript Object Notation) is a lightweight data-interchange format with a focus on human readability.

which is an unordered collection of key/value pairs. Note that a value can, in turn, be a collection of key/values. The value of the `type` key is a string and denotes the type of message found in `data`:

- CC - The create container message that creates a secure computing environment in the cloud;
- DC - The destroy container message destroys an existing computing environment;
- CN - The configure components message enables the launching/stopping of ROS nodes, the setting/removal of parameters in the ROS parameter server, and the adding/removal of *Interfaces*;
- CX - The configure connections message enables the connection/disconnection of *Interfaces*;
- DM - The data messages are used to send/receive any kind of messages to/from application nodes (for more examples see Sec. 2.4);
- ST - Status messages are pushed from Rapyuta to the robot; and
- ER - Error messages are also pushed from Rapyuta to the robot.

Handling Large Binary Messages

The WebSocket interface supports transportation of binary blobs and, for some types of data, it is better to transport them as a binary blob instead of using their corresponding ROS message type encoded as a JSON string. For example, the RoboEarth logo (RGBA, 842×595), if transported as PNG (lossless data compression), takes 18 kB in bandwidth but uses approximately 2.0 MB when transported as a serialized ROS message. Converting the ROS message into a JSON string would result in an even larger message size.

To exploit this method of transportation, special converters between the binary format and the corresponding ROS message must be provided on the Rapyuta's *interface* side. Rapyuta provides a default PNG-to-sensor_msgs/Image converter as an example of how to build new converters.

When sending a binary message, first a standard data message is sent as a JSON string with a reference to the binary blob that will follow. The message is a DM type message having a `data` key with value:

```
"iTag"  : "converter_modifyImage",
"type"  : "sensor_msgs/Image",
"msgID" : "msgID_0",
"msg*"  : "f9612e9b3c7945ef8643f9f590f0033a"
```

The '*' in the last line indicates that the value/resource will follow as a binary blob with the given ID as header. Note that the ID must be unique only within the current connection.

Communication with RoboEarth

By default, every container has a `py_re_comm` node running inside it. This ROS node exposes the RoboEarth repository by providing services to download, upload, update, delete, and query action recipes, object models, and environments stored in the RoboEarth repository. Since the RoboEarth repository is also typically hosted in the same data center, all applications running on Rapyuta have high bandwidth access to the data, unlike applications running on board the robot.

Virtual Networks

As described in Sec. 2.3, processes running in different computing environments (containers) communicate through Rapyuta's internal communication protocol built on top of ROS. However, some applications that are distributed over multiple containers, may require a less abstracted version of the network to use different protocols such as Open MPI [24]. Containers within a common host could communicate using the LXC bridge, which is the default network interface for containers. However, the LXC bridges of different host machines cannot be connected directly. Therefore, the current version of Rapyuta includes the functionality to create a virtual network with an arbitrary topology between containers that belong to a specific user. The virtual network is realized using Open vSwitch [25], which is connected to an additional network interface of the container. See benchmarking results in Sec. 2.5 for comparisons between the virtual networks and Rapyuta's internal communication protocol.

2.4 Deployment

The core components and communication protocols described in the previous sections can be combined in different ways to meet the specifications of a robotic scenario. This section presents three typical use cases, a basic example of the communication process and some useful tools.

Use cases

Figure 2.7 shows the standard use case where the four task sets are split up into the four processes (the *Master* process, *RobotEndpoint* process, *EnvironmentEndpoint* process, and the *Container* process), and combined with interconnected computing environments to build a PaaS framework. The *Master* process runs on a single dedicated machine. Other machines each run both a *RobotEndpoint* and a *Container* process. The two task sets are run separately, since the *Container* process requires super user privileges to start and stop containers which could pose a severe security risk when combined with the open

See `http://github.com/IDSCETHZurich/re_comm_core` for more details.

See `http://rapyuta.org/install` and `http://rapyuta.org/usage` for more details on the setup and usage of the standard use case.

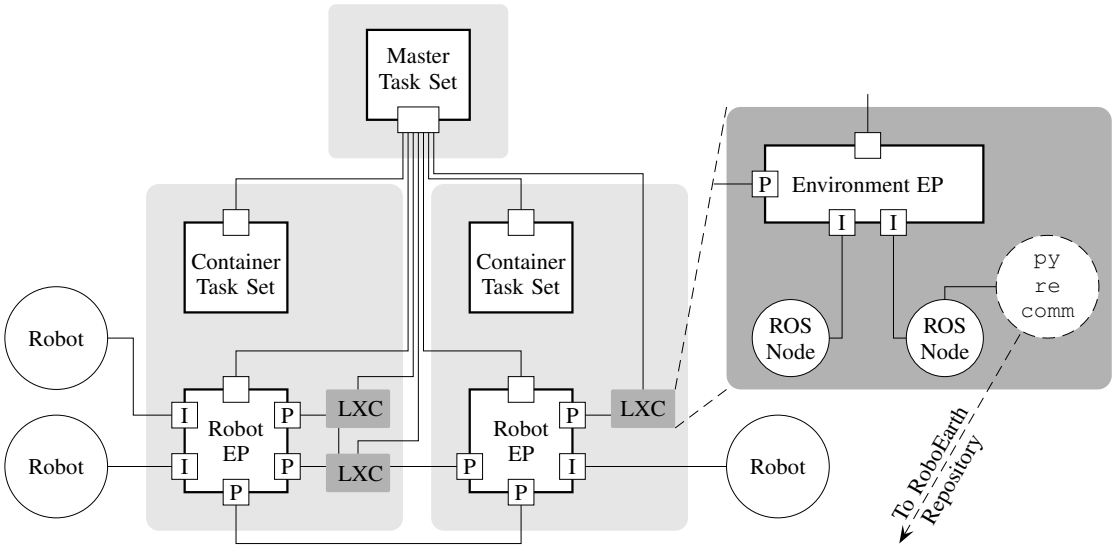


Figure 2.7: Use Case 1: The typical use case of Rapyuta processes deployed on three machines (light-gray blocks) to build a PaaS framework with interconnected computing environments (LXC, dark-gray blocks). Here the *Master* task set runs as a single process on one of the machines and the other two machines are used to deploy containers. Inside each machine that hosts containers, the robot task set runs as a single process, and inside each container the environment task set runs as a single process. The computing environment denoted by LXC (Linux Containers) is enlarged in the right side of the figure. Note that the dashed arrow from the `py_re_comm` node denotes the connection to the RoboEarth knowledge repository within the same cluster/data center, thus providing a high bandwidth access.

accessible *RobotEndpoint* process. The fourth process, the *EnvironmentEndpoint* process, is running inside every computing environment. Note that this configuration allows all three elastic computing models to be deployed for cloud robotics, as proposed in [16]; the peer-based, proxy-based, and the clone-based model. From an administrative point of view, the standard use case can be deployed in the following ways:

- Private cloud: Rapyuta, the applications running on it, and the robots belong to a single entity. This is better suited for some commercial entities where trust and security is the highest concern.
- Software-as-a-Service: Rapyuta and the applications running on it belong to a single entity, and several users connect and use the applications. This allows the single entity to better protect its intellectual property, keep the software up to date, and provide better support.
- Platform-as-a-Service: Here, only the Rapyuta platform is managed by a single entity, while a community of developers develop and share/host the applications. In addition to the advantages stated above, this allows for easy benchmarking and ranking of various solutions to robotics.

The second use case is an extreme case of the standard use case where everything runs on a single machine with one container. This mimics a rosbridge [15] system and can be used as a sandbox to develop cloud robotics applications and investigate latencies.

Finally, the third use case presented in Fig. 2.8 shows how to set up a network of robots using the *RobotEndpoint* and *Master* processes. Although Fig. 2.8 shows a single machine, multiple machines with interconnected *Endpoint* processes are also feasible.

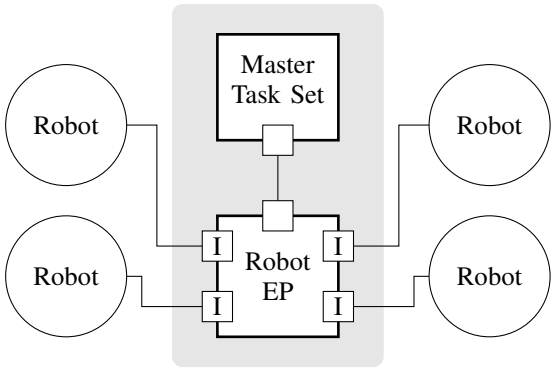


Figure 2.8: Use Case 2: Process configuration for setting up a network of robots running a *RobotEndpoint* and *Master* process in a single machine (light-gray block).

Note that the machines mentioned in all three use cases (light-gray blocks in Figs. 2.7 and 2.8) can also be instances of an IaaS [4] provider such as Amazon EC2 [26] or Rackspace [27].

Basic Communication Example

In order to illustrate the usage and communication protocols, this subsection provides a simple example of a communication process with Rapyuta's standard use case setup (See Fig. 2.7). Here a Roomba vacuum cleaning robot with a wireless connection uses Rapyuta to record/log its 2D pose. The communication takes place in the following order:

Initialization The first step for the Roomba is to contact the process running the *Master* task set using the user ID `roombaOwner` to get the address of a *RobotEndpoint*. This is done with the following HTTP request:

```
http://[domain]:[port]?userID=roombaOwner&version=[version]
```

A *RobotEndpoint* is selected of the available *Endpoints* and the *Endpoint*'s URL is returned to the Roomba as a JSON encoded response.

```
{
  "url":"ws://[domain]:[port]/"
}
```

In the second step of initialization, Roomba makes a connection using the received URL of the assigned robot *Endpoint*, registers using the robot ID `roomba`, and logs in using the API key `secret`. This is done with the following HTTP request:

```
ws://[domain]:[port]/?userID=roombaOwner&robotID=roomba&key=secret
```

Container Creation The Roomba creates a computing environment and tags it with a CC-type message having a `data` key with value:

```
"containerTag" : "roombaClone"
```

Note that the tag must be unique within the robots that use the same user ID. Container creation also automatically starts the necessary Rapyuta processes inside the container.

Configure Nodes The Roomba launches the logging node (posRecorder.py) and starts two *Interfaces* with tags using a CN-type message having a `data` key with value:

```
"addNodes" : [{
    "containerTag" : "roombaClone",
    "nodeTag"      : "positionRecorder",
    "pkg"          : "testPkg",
    "exe"          : "posRecorder.py"
  }],
"addInterfaces" : [{
    "endpointTag"   : "roomba",
    "interfaceTag"  : "pos",
    "interfaceType" : "SubscriberConverter",
    "className"     : "geometry_msgs/Pose2D"
  }, {
    "endpointTag"   : "roombaClone",
    "interfaceTag"  : "pos",
    "interfaceType" : "PublisherInterface",
    "className"     : "geometry_msgs/Pose2D",
    "addr"          : "/posPub"
  }]
```

Note that the above complex message can be split into multiple messages that launches the node and start *Interfaces* separately.

Binding Interfaces Before the Roomba can use the added node the two *Interfaces* must be connected. This is achieved with a CX-type message having a `data` key with value:

```
"connect" : [{
    "tagA" : "roomba/pos",
    "tagB" : "roombaClone/pos"
  }]
```

Data Finally, the Roomba starts sending the data message that contains the 2D pose information, i.e., a DM-type message having a `data` key with value:

```
"iTag"  : "pos",
"type"  : "geometry_msgs/Pose2D",
"msgID" : "id",
"msg"   : {
    "x" : 3.57,
    "y" : -44.5,
    "theta" : 0.581
  }
```

This data message (ASCII JSON) is converted to a ROS message at the *Interface* `roomba/pos` and is sent to the *Interface* `roombaClone/pos`. The *Interface* `roombaClone/pos` then transfers the message to the `posRecorder.py` node via the ROS environment. Now, by adding *Interfaces* of type `PublisherConverter` and `SubscriberInterface`, a topic can also be transferred from the nodes running in `RoombaClone` to the robot `roomba`.

Tools

Managing a cloud-based environment is a complex and cumbersome task. To simplify the management of Rapyuta, a console client for administrators and users is provided. This tool allows users to monitor Rapyuta's components based on their privileges and to interact with Rapyuta similar to the external protocol described in Sec. 2.3.

Setting up Rapyuta can be the first and the biggest hurdle for a beginner. To address this issue, Rapyuta provides a provisioning script for users who want to setup and use Rapyuta in their own hardware. The script sets up the full system, including the networking, containers, and their file system. This provisioning script is compatible with almost all of the recent Ubuntu (12.04,12.10,13.04) and ROS (Fuerte, Groovy) variants. For Amazon EC2 users, Rapyuta provides an Amazon Machine Image (AMI) with the latest stable version, which they can copy and start using within minutes.

Administrative Tools

Managing a cloud-based environment is a complex and cumbersome task. The computing environments often lie distributed across several hundred machines. Each machine in turn has numerous parameters and components such as containers, endpoints, interfaces, nodes and processes to name a few. Rapyuta provides a console client for administrators and users to monitor Rapyuta's components based on their privileges and to interact with them similar to the external protocol described in Sec. 2.3.

Setup Tools

Setting up Rapyuta, a relatively complex system in the robotics domain, was the first and the biggest hurdle for a beginner . Rapyuta now provides two relatively easy ways to setup. For users who want to use Rapyuta in their own hardware can use the provisioning scripts

For more details on the console client see `http://rapyuta.org/Console`
For more details on the setup tools for installation see `http://rapyuta.org/Install`
For more details on the console client see `http://rapyuta.org/Console`

to setup the full system including the networking, containers, and theirfile system. Note that there provisioning scripts cover all most all of the recent Ubuntu and ROS variants. For Amazon EC2 users, Rapyuta provides an Amazon Machine Image (AMI) with the latest stable version, which they can copy and start using within minutes.

2.5 Performance and Benchmarking

In this section we provide various performance measures of Rapyuta under different configurations and provide benchmarking results with rosbridge. All experiments were conducted by measuring the round-trip times (RTTs) of different sized messages between two processes. Note that all experiments are a variation of: where the two processes were running, their communication route, and the transport mechanism. A topic-based and a service-based transport mechanism were used. For each experiment, 25 message sizes (log-uniformly distributed between 10 B and 10 MB) were selected and, for each size, 20 samples were measured. All experiments, except for the remote process/robot case, were performed on a machine instance in Amazon's Ireland data center. A machine located at ETH Zurich, Switzerland was used to replicate the remote process/robot.

In addition to the transmission delays (message size/bandwidth), round-trip times also contain other factors such as queuing time, processing time and propagation delay (distance/propagation speed). For smaller messages, the influence of these other factors diminish the effects of transmission delays, giving a flat RTT for message of sizes up to 10 kB.

For interpretation and comparison, note that a `tf`-typed message that contains the relationship between multiple coordinate frames of the robot shown in Fig. 3.3 is around 100 B; whereas and an RGB-D frame, which contains a PNG compressed RGB and depth images in VGA resolution, is around 500 kB.

Rapyuta Core

In this part, we compare RTTs for all the core data routes of the standard use case as shown in Fig. 2.7. The results of the experiments are shown in Fig. 2.9 for the topic-based transport mechanism and Fig. 2.10 for the service-based transport mechanism. Since a new connection must be established for every service call that is made, services show a higher latency than topics. However, the trends with respect to RTTs of different routes remain the same under both transport mechanisms.

The results in Figs. 2.9 and 2.10 show:

- Communication with an external process (R2C) is the biggest constraint of Rapyuta's throughput.
- The difference between containers running in the same machine (C2C-1) and different machines (C2C-2), resulting from the iptables and port forwarding overheads,

For more details on the setup tools for installation see `http://rapyuta.org/Install`

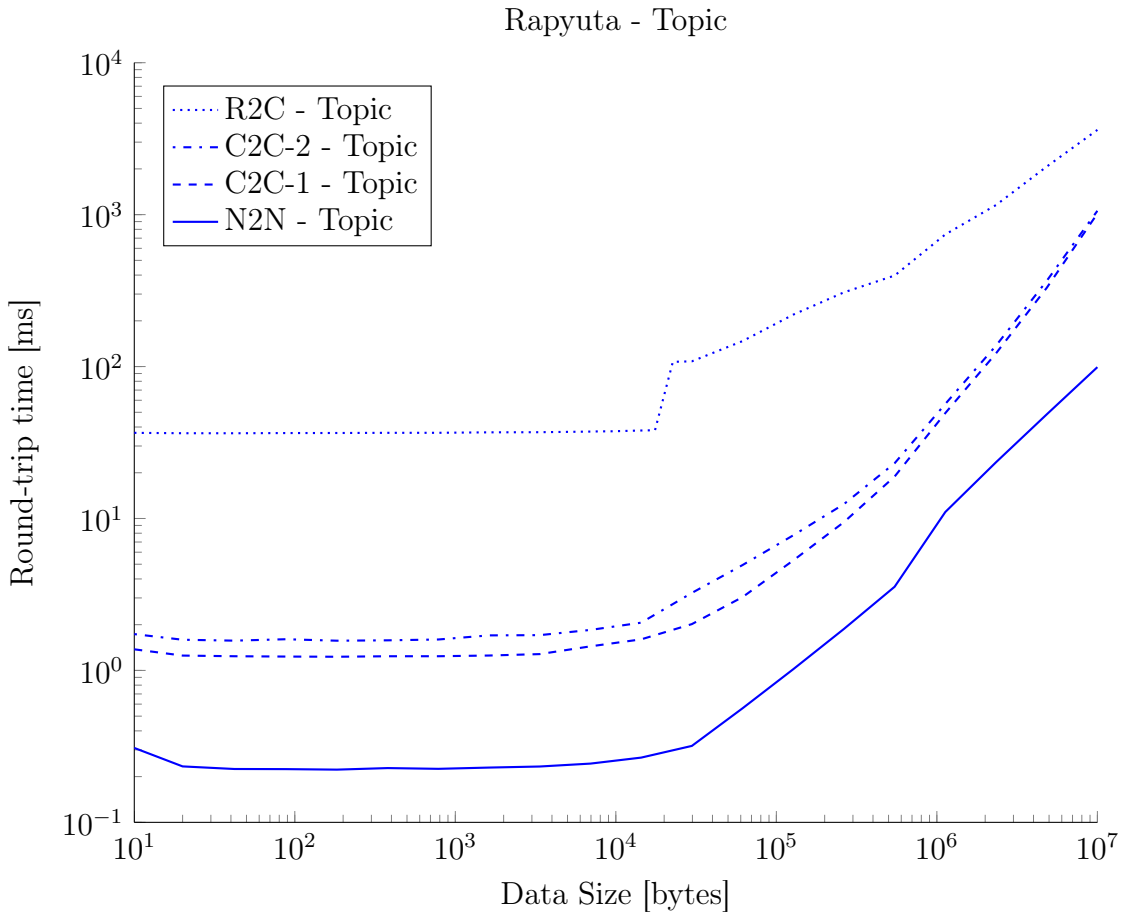


Figure 2.9: RTTs for different data routes in a standard use case (see Fig. 2.7) under the topic transport mechanism. N2N denotes the communication of two processes (nodes) within the same ROS environment inside a container; C2C denotes two processes in two different containers, where for C2C-1 the containers are running on the same machine and for C2C-2 on different host machines. Finally R2C denotes the communication between a remote process and a process running inside Rapyuta's containers.

can be neglected in comparison to the difference to the communication between two nodes in the same ROS environment (N2N).

- Rapyuta introduces an overhead of < 2 ms for topics and 5 ms for services for data sizes up to 10 kB (see Fig. 2.9), which can be seen from the differences between C2C-1 and N2N.

rosbridge

Here we compare Rapyuta with rosbridge with respect to RTTs. Figures 2.11 and 2.12 show Round-trip times (RTTs) for communication with an external non-Rapyuta process that runs on the same machine where the framework (Rapyuta/rosbridge) is running, and on a remote machine respectively. In both cases, communication speeds remain almost constant for small message sizes up to 10 kB, and increase linearly for larger sizes.

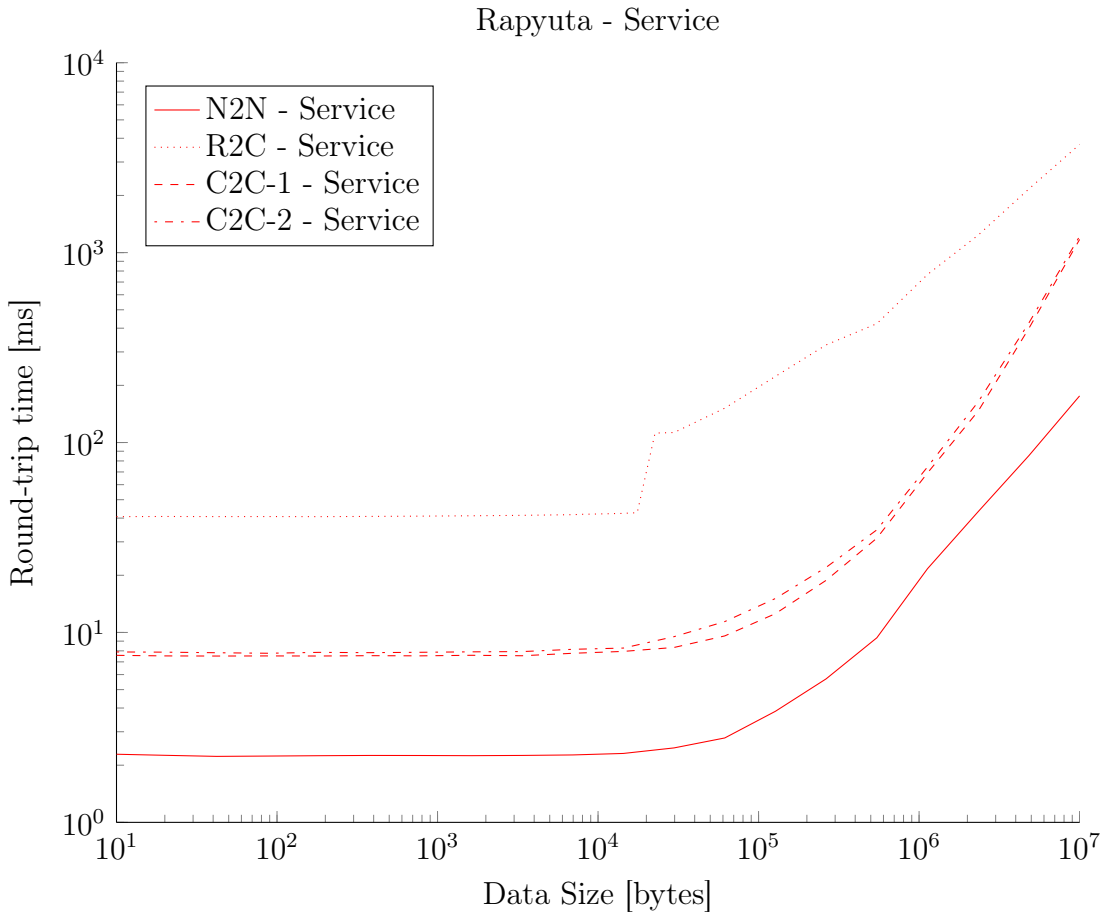


Figure 2.10: RTTs for different data routes in a standard use case (see Fig. 2.7) under the service transport mechanism.

For external processes running on the same machine (Fig. 2.11), RTTs are dominated by the queuing and processing times. For small message sizes, both Services and Topics show lower RTTs for rosbridge compared to Rapyuta. Conversely, Rapyuta shows lower RTTs than rosbridge for larger message sizes. Note that while optimization allows Rapyuta to improve communication speeds for larger messages, the additional inter-process communication hops required in a multiple computing environment slow down Rapyuta's speed when delivering small messages. For remote external processes (Fig. 2.12), RTTs are dominated by the transmission delays of the Internet connection. These transmission delays diminish rosbridge's advantage over small message, and results in a similar performance between Rapyuta and rosbridge. For larger messages ($10 - 300$ kB) rosbridge RTTs are lower for topics compared to Rapyuta, but higher for services. For message sizes larger than 500 kB, Rapyuta offers lower RTTs. Differences in the WebSocket libraries used by rosbridge and Rapyuta, and C optimizations done for Rapyuta's AutoBahn WebSocket library account for these RTT discrepancies.

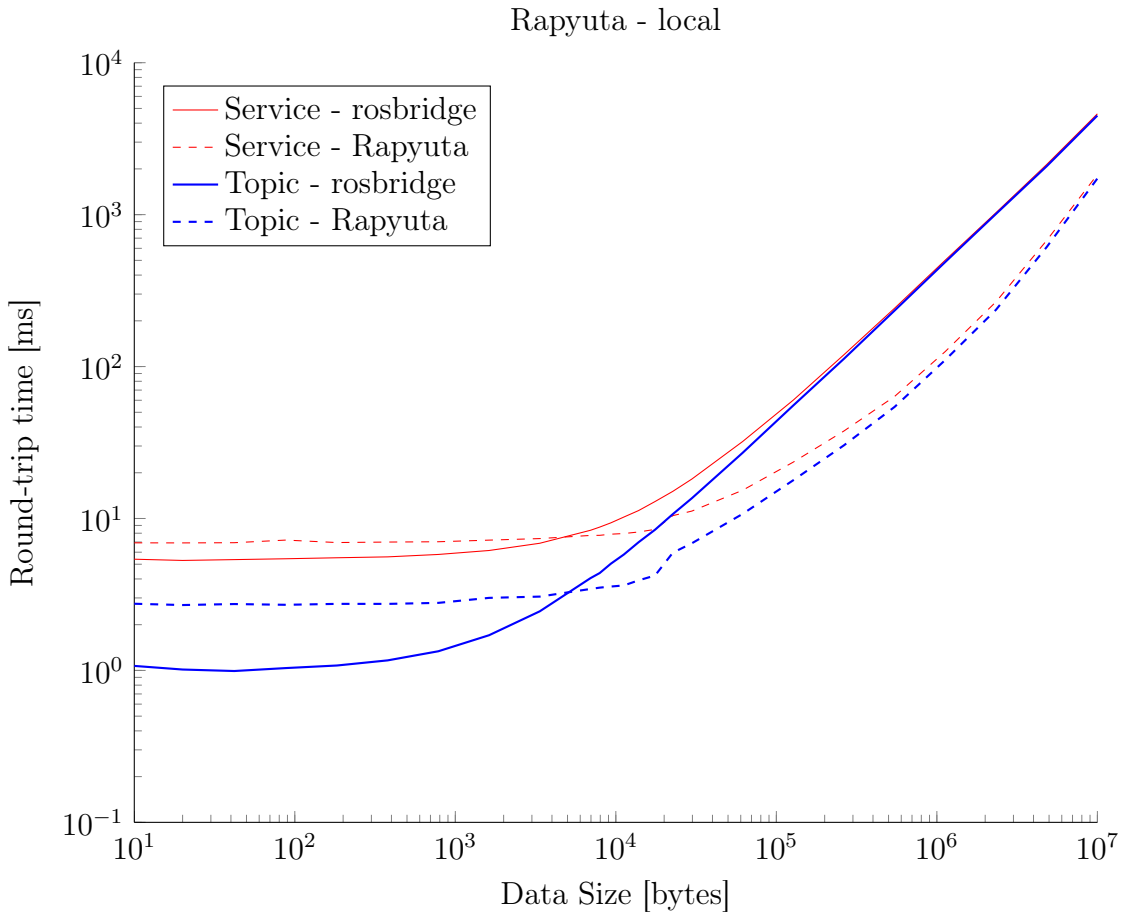


Figure 2.11: RTTs with the external process running on the same host machine that is running Rapyuta/rosbridge.

Virtual Network-based Internal Communications

Here we compare the virtual networks discussed in 2.3 to Rapyuta's internal communication protocol. Note that all virtual network-based experiments between two containers had one single ROS master managing the communications as opposed to one ROS master per container in a typical Rapyuta use case. Figs. 2.13 and 2.14 compare the communication between two containers under the virtual network using ROS and Rapyuta's internal communication protocol. Except for LXC bridge's artifact for service calls with larger messages, the virtual networks are superior to Rapyuta's internal communication protocol. Note that this comes at the cost of loosing security and encapsulation, and is thus only allowed within a single user's computing environments.

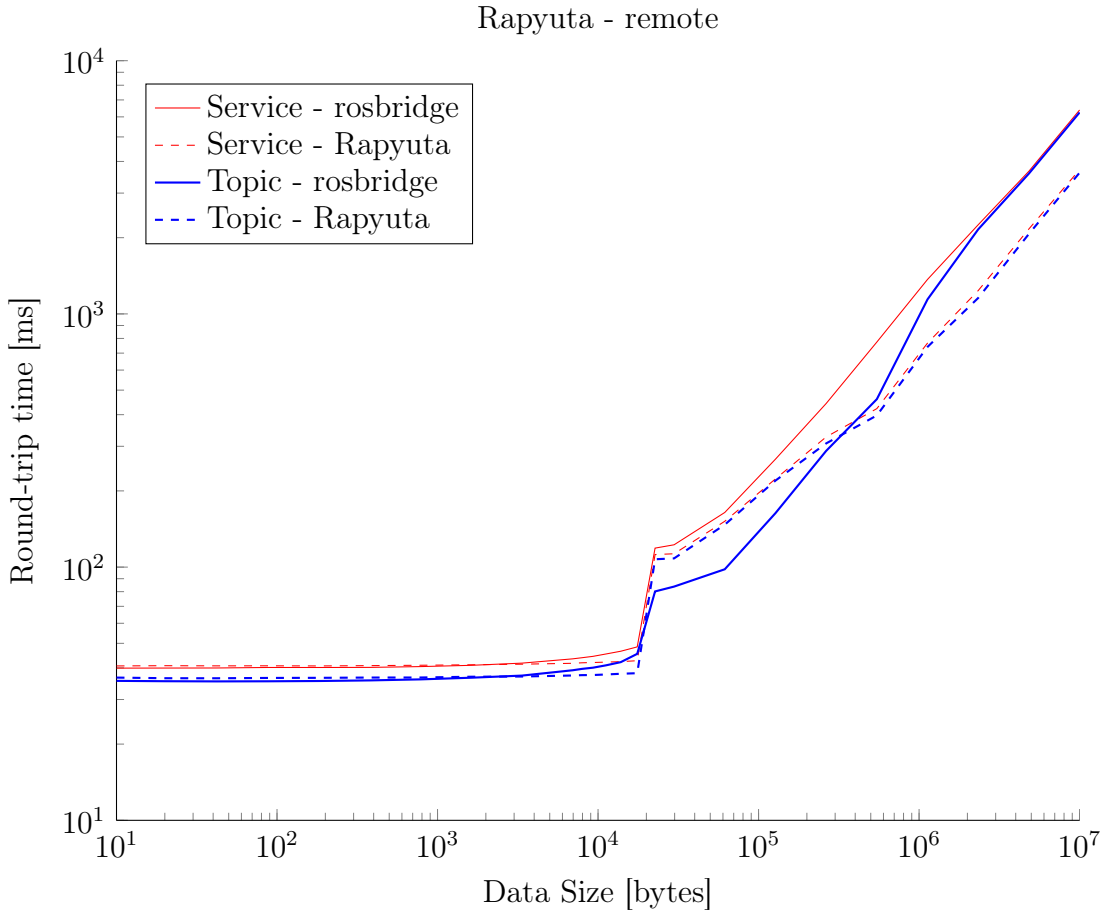


Figure 2.12: RTTs with a remote external process running at ETH Zurich, Switzerland while Rapyuta and rosbridge ran on Amazon's Ireland data center.

2.6 Demonstratros

In addition to the tutorial applications that come with the source code, we also developed two typical robotic applications to highlight various aspects of Rapyuta.

Demonstrator: Cloud-based Mapping

As a first proof-of-concept demonstrator, we implemented a cloud-based collaborative mapping service in Rapyuta. The robots shown in Fig. 3.3 consist mainly of off-the-shelf components. They use the iRobot Create as their base, and this differential drive base provides a serial interface to send control commands and receive sensor information. PrimeSense CARMIN 1.08 is used for the RGB-D sensing. A 48×52 mm embedded board with a smartphone-class multi-core ARM processor is used for onboard computation. The embedded board runs a standard Linux operating system and connects to the cloud through a dual-band USB wireless device. This hardware setup was used with different software configurations to highlight different aspects of the challenges and opportunities.

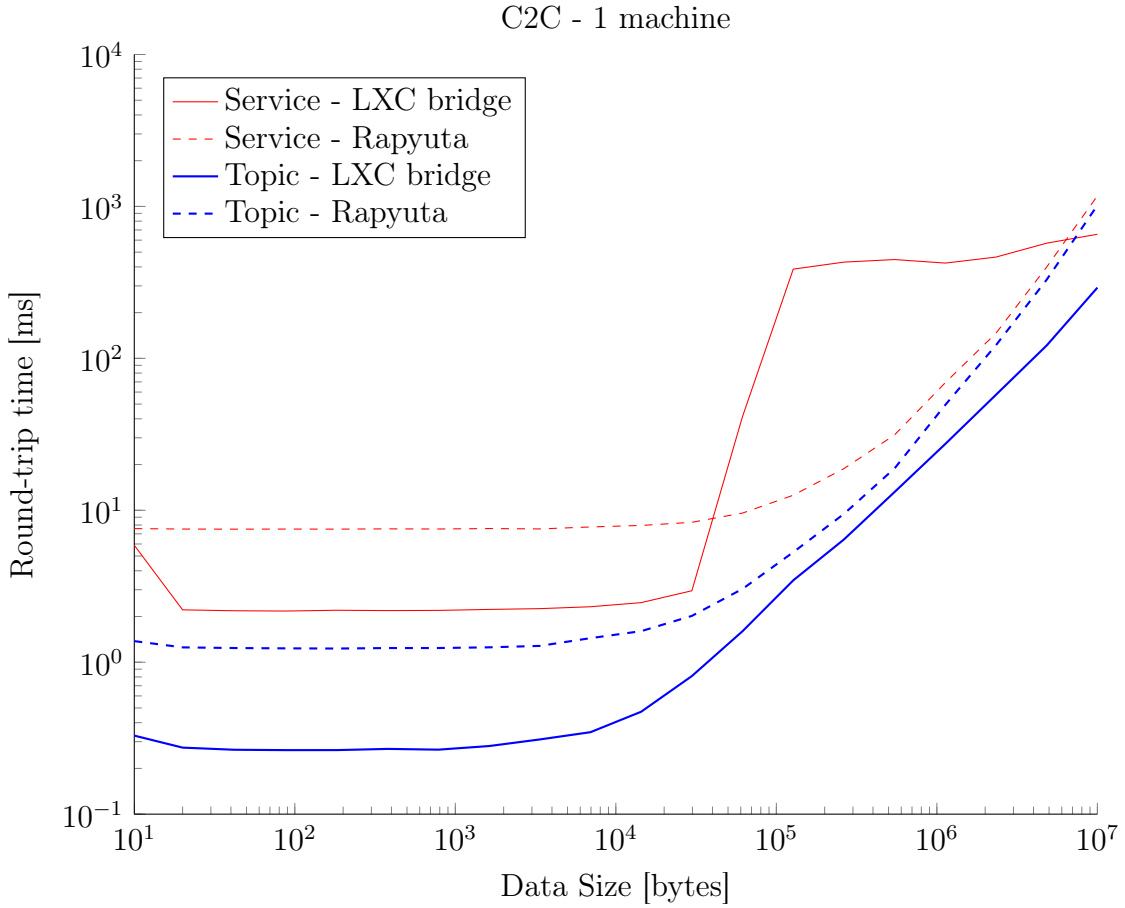


Figure 2.13: RTTs with LXC bridge-based virtual networks and Rapyuta. The two containers that were running the communication processes were hosted by a single host machine.

Complete Offloading In this setup almost all of the processing was moved Rapyuta and the only computation on board was the compression of RGB-D data. At ROSCON 2013 (Stuttgart, Germany) this setup was demonstrated by doing a frame-by-frame compression onboard and doing all the rest in Rapyuta at Amazon's data center in Ireland, which resulted in a 5 MB/s throughput at 30Hz in QVGA resolution. This setup demonstrated the practical limits of off-the-shelf wireless devices and helped us stress test Rapyuta in terms of throughput. Under this frame-by-frame compression, any resolution higher than the QVGA resulted in dropped frames due to the bandwidth limitation. For more effective ways of sending visual and depth information see Sec.2.6 below and Sec.2.6.

Local Visual Odometry In this setup, a dense visual odometry algorithm [28] was run on board the local system, and only the RGB-D key-frames were sent to Rapyuta for the global optimization. This setup used a bandwidth of 300-500 kB/s with QVGA-

The source code of the dense visual odometry-based mapping demonstrator is available at `http://github.com/IDSCETHZurich/rapyuta-mapping` under the Apache 2.0 license.

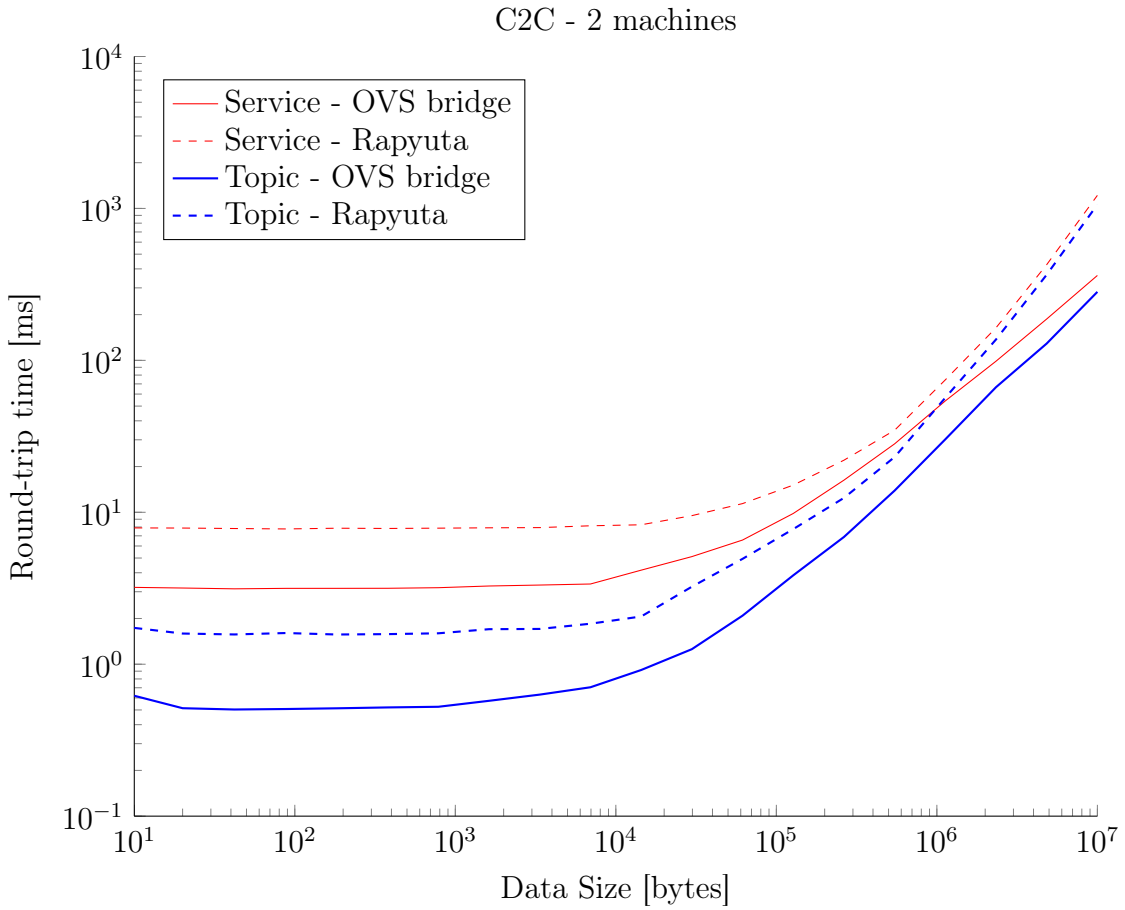


Figure 2.14: RTTs with Open vSwitch [25]-based virtual networks and Rapyuta. The two containers that were running the communication processes were hosted separately on two host machines.

resolution key frames. This setup demonstrated a good trade off between the data rates and computation given the wireless speed, and the robots' speed and available computation. Furthermore, this setup also had a component running on Rapyuta that directed the robot to its next exploration point based on predefined set points. This component can be extended to automatically provide set points based on the map being built.

Collaborative Mapping In this setup, multiple robots were used to collaboratively build a 3D map of an environment. This setup demonstrated that the cloud can serve not only as a computational resource, but also as a common medium for collaborative tasks. A 3D model of an environment created by this method is shown in Fig. 3.1.

Demonstrator: Dense Mapping with Rapyuta

As a second demonstrator we implemented a dense mapping service, which compressed and streamed all of the depth sensor data at VGA resolution to the GPU processes running on Rapyuta. This demonstrator used the point cloud library's (PCL) [29] imple-

Figure 2.15: The two low-cost (∼ 600$) robots used in our demonstrators: Each robot consists mainly of a differential drive base (iRobot Create), an RGB-D sensor (PrimeSense), an ARM-based single board computer (ODROID-U2), and a dual band USB wireless device.

mentation of the KinectFusion [30] algorithm, which does not rely on keyframes as in the previous demonstrator, but utilizes all images and all of their pixels to reconstruct the map. This prevents the mapping algorithm from being split up (as suggested previously) to reduce the necessary bandwidth, and is therefore used as an example for an application where a large amount of data must be exchanged. Libav [31] (an open-source video and audio processing library) was used to compress the RGB-D stream as two separate video streams. For this demonstrator we compressed the color images using the common compression scheme h264. For the depth images we used the FFV1 video codec, since FFV1 natively supports the lossless compression of a 16-bit monochrome image stream, whereas h264 does not support this type of pixel format.

Figure 2.16 shows a sample frame of the KinectFusion output. The bandwidth usage of the RGB images in this particular case was 1.1 MB/s for the dynamic scene and 800 kB/s for a static scene. Compared to 27.6 MB/s (which is the bandwidth requirement of raw RGB images) the compression results in a reduction of 96%. Similarly, the bandwidth required for the depth images was 1.6 MB/s for a dynamic scene and 1.4 MB/s for a static

The source code of the depth mapping demonstrator can be found at `http://github.com/IDSCETHZurich/rapyuta-kinfu`. under the Apache 2.0 license.

Figure 2.16: A frame of the KinectFusion output running on an Amazon GPU instance in Ireland.

scene. This resulted in a reduction of 92%, compared to the 18.4 MB/s requirement of the raw depth images. Note that all values are for a frame rate of 30 frames per second.

The second notable aspect of this demonstrator is that it uses the GPU to process the incoming data in a timely manner. This allow us to demonstrate that although the algorithm runs isolated in a virtual machine, the container is flexible enough to allow direct access to the GPU, a hardware component of the host machine.

2.7 Conclusion and Outlook

In this paper we described the design, implementation, benchmarking results, and the first demonstrators of Rapyuta, a PaaS framework for robots. Rapyuta, based on an elastic computing model, dynamically allocates secured computing environments for robots.

We showed how the computing environments and the communications protocols allow robots to easily offload their computation to the cloud. We also described how Rapyuta's computing environments can be interconnected to share specific resources with other

environments, making it a suitable framework for multi-robot control and coordination.

Our choice of communication protocols were explained, and an example was provided to clarify the different types of messages and to show how they work together. With respect to communication, we also provided some benchmarking results for different protocols.

Next, we showed the flexibility of Rapyuta's modular design by giving three specific use cases as a guide. Finally, two example robotic demonstrators were presented to highlight various aspects of Rapyuta and cloud robotics in general. These demonstrators also provided practical examples on how to handle application-specific tradeoffs between available onboard computation, the application's bandwidth and real-time requirements, the reliability of communication with the cloud, and the available cloud infrastructure.

Together with the RoboEarth Knowledge Repository, Rapyuta provides an appropriate cloud robotics platform that has the potential to improve robotics in the following ways:

- provides massively-parallel computation on demand for sample-based statistical modelling and motion planning [32],
- leverages the cloud as a real-time communication medium for collaborative task performance and information sharing,
- serves as a global repository to store and share object models, environment maps, and actions recipes between various robotic platforms; enabling life-long learning,
- Robotic Application-as-a-Service: eliminates setup and update overhead for the end users, serves as a better model for protecting intellectual property for the developers, and functions as a common platform to benchmark different algorithms and solutions, and
- allows humans to monitor or intervene and help robots when they are lost; this not only makes the robotic system more robust but also provides a lot of labelled data to learn from as an intermediary step before humans are taken out of the loop.

Many more applications can be found in the field of intelligent transportation, environmental monitoring, smart homes and defence [16].

Acknowledgment

This research was funded by the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement no. 248942 RoboEarth and was supported by AWS (Amazon Web Services) in Education Grant award. The authors would like to express their gratitude towards Vladyslav Usenko (TUM) for help building the mapping demonstrator, Dhananjay Sathe and Mayank Singh for their help with the software development, Carolina Flores and Christine Waibel for helping with the promotional video, Matei Ciocarlie

(Willow Garage) for the excellent feedback on the draft, and all RoboEarth colleagues for their support and feedback.

References

[1] R. Arumugam, V. R. Enti, K. Baskaran, and A. S. Kumar, "DAvinCi: A cloud computing framework for service robots," in *Proc. IEEE Int. Conf. Robotics and Automation*. IEEE, May 2010, pp. 3084–3089.

[2] B. Kehoe, D. Berenson, and K. Goldberg, "Toward cloud-based grasping with uncertainty in shape: Estimating lower bounds on achieving force closure with zero-slip push grasps." in *Proc. IEEE Int. Conf. Robotics and Automation*. IEEE, May 2012, pp. 576–583.

[3] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft, "Roboearth," *Robotics Automation Mag., IEEE*, vol. 18, no. 2, pp. 69 –82, june 2011.

[4] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Special Publication 800-145, 2011, available http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[5] Google, "Google App Engine," 2008. [Online]. Available: https://developers.google.com/appengine/

[6] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[7] J. Lindenbaum, A. Wiggins, and O. Henry, "Heroku," 2007. [Online]. Available: http://www.heroku.com/

[8] VMware, Inc., "Cloud Foundry," 2013. [Online]. Available: http://www.cloudfoundry.com/

[9] Red Hat, Inc, "OpenShift," 2013. [Online]. Available: https://openshift.com/

[10] K. Goldberg and R. Siegwart, Eds., *Beyond webcams: an introduction to online robots*. Cambridge, MA, USA: MIT Press, 2002.

[11] M. Inaba, S. Kagami, F. Kanehiro, Y. Hoshino, and H. Inoue, "A platform for robotics research based on the remote-brained robot approach." *I. J. Robotic Res.*, vol. 19, no. 10, pp. 933–954, 2000.

[12] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[13] K. Kamei, S. Nishio, N. Hagita, and M. Sato, "Cloud Networked Robotics," *Network, IEEE*, vol. 26, no. 3, pp. 28–34, May-June 2012.

[14] M. Sato, K. Kamei, S. Nishio, and N. Hagita, "The ubiquitous network robot platform: Common platform for continuous daily robotic services," in *System Integration (SII), 2011 IEEE/SICE Int. Symp.*, Dec 2011, pp. 318 –323.

[15] G. T. Jay, "brown_remotelab: rosbridge," 2012. [Online]. Available: http://www.rosbridge.org/

[16] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *Network, IEEE*, vol. 26, no. 3, pp. 21–28, May-June 2012.

[17] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses," *AI Magazine*, vol. 29, no. 1, pp. 9–20, 2008.

[18] "Linux Containers," 2012. [Online]. Available: http://lxc.sourceforge.net/

[19] "chroot, Linux programmer's manual," 2012. [Online]. Available: http://www.kernel.org/doc/man-pages/online/pages/man2/chroot.2.html

[20] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–10.

[21] G. Lefkowitz, "Twisted," 2012. [Online]. Available: http://twistedmatrix.com/

[22] I. Fette and A. Melnikov, "The WebSocket Protocol, RFC 6455," 2011. [Online]. Available: http://tools.ietf.org/html/rfc6455

[23] Tavendo GmbH, "Autobahn WebSockets," 2012. [Online]. Available: http://autobahn.ws/

[24] Community Project, "Open MPI: Open Source High Performance Computing," 2013. [Online]. Available: http://www.open-mpi.org/

[25] ——, "Open vSwitch," 2013. [Online]. Available: http://openvswitch.org/

[26] Amazon.com Inc., "Amazon Elastic Compute Cloud," 2012. [Online]. Available: http://aws.amazon.com/ec2/am

[27] Rackspace US, Inc., "The Rackspace Open Cloud," 2012. [Online]. Available: http://www.rackspace.com/

[28] F. Steinbrucker, J. Sturm, and D. Cremers, "Real-time visual odometry from dense RGB-D images," in *ICCV Computer Vision Workshop*, 2011, pp. 719—–722.

[29] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[30] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, 2011, pp. 127–136.

[31] "libav," 2013. [Online]. Available: http://libav.org/

[32] USA Robotics VO, "A Roadmap for U.S. Robotics, From Internet to Robotics, 2013 Edition," 2013.

3

Cloud-based Collaborative 3D Mapping with Low-Cost Robots

3.1 Introduction

The past decade has seen the first successful, large-scale use of mobile robots. However, a large proportion of these robots continue to either use simple control strategies (e.g. robot vacuum cleaners) or be remotely operated by humans (e.g. drones, telepresence robots). A primary reason for the lack of more complex algorithms in such systems is the cost (both direct and indirect) of onboard computation and storage.

The rapid progress of wireless technologies and the availability of commercial data centers, with high-bandwidth connections and highly scalable computation, storage, and communication infrastructures ('the cloud' [1]) may allow robots to overcome many of the current bottlenecks. Currently, several frameworks [2], [3], [4], [5] and robotic applications [6], [7] are being developed to exploit the cloud's potential for creating light, fast, and intelligent low-cost robots.

In this paper, we focus on using the cloud for mapping and localization – two of the most important tasks for any mobile robot. The process of simultaneously building a map and localizing a robot, also known as Simultaneous Localization and Mapping (SLAM), has been a research topic for many years and many SLAM algorithms have been proposed. Although the algorithms are increasing in precision, they require substantial onboard computation and often become infeasible when used for making larger maps over a long period of time. Furthermore, running everything locally also limits the potential for collaborative mapping.

This paper is submitted to the *IEEE Transactions on Automation Science and Engineering*, March 2014.

(a) Top view

(b) Side view with a photo taken in a similar perspective.

Figure 3.1: A point cloud map of a room at ETH Zurich built in real-time by the two robots shown in Fig. 3.3. The individual maps generated by the two robots are merged and optimized by processes running on a datacenter in Ireland. The robots are re-localized and the robot models are overlaid in the merged map [2].

A cloud-based parallel implementation of Fast-SLAM [8] was presented in [4] and showed a significant reduction in computation time. In this work, the authors presented a cloud infrastructure based on Hadoop [9] and received data from the robot using a common Robot Operating System (ROS) [10] master that managed all communications. Similar to [4], authors of [11] proposed a collaborative mapping framework where they moved the computationally intensive bundle adjustment process of the Parallel Tracking and Mapping (PTAM) [12] algorithm to a high performance server connected to the client computer. In addition to the above robotic scenarios, the Kinect@Home project [13] aims to develop a collection of RGB-D datasets through the use of crowdsourcing, by allowing any user with a Kinect and an appropriate web browser plugin to scan their environment. Once the dataset is uploaded, Kinect@Home performs a batch optimization and generates a 3D representation of the map for the user in the web browser.

Matterport [14] is now developing a commercial system with custom cameras (similar to Kinect@Home), with the goal of making it easy for anyone to create 3D images of real-world spaces and share them online. Several centralized collaborative approaches that have the potential to run in a decentralized manner also exist. A 2D mapping system using manifold representation was introduced in [15], where the problem of map optimization and merging maps from different robots has been discussed. However, loop closure and map merging were only possible when another robot was recognized visually. In [16] the authors present a collaborative visual SLAM system for dynamic environments that is capable of tracking camera pose over time and deciding if some of the cameras observe the same scene; information is combined into groups that run the tracking together. More recently, several visual-inertial odometry systems [17], including Google's Project Tango [18] that runs on a custom cellphone with specialized hardware, has shown superior accuracy and consistency over the other approaches. But scalability, global optimization, and map merging remains open in the above mentioned visual-inertial systems.

This paper shows that low-cost robot platforms with a smartphone-class processor and a wireless connection are able to collaboratively map relatively large environments at quality levels comparable to the current SLAM methods. Furthermore, this paper shows a scalable approach to map optimization, storage, and merging of maps from different sources.

The main contributions of this paper are:

- Open source parallel implementation of dense visual odometry on a smartphone-class ARM multi-core CPU
- A novel cloud-based SLAM architecture and protocol, which significantly reduces the bandwidth usage
- Techniques for parallel map optimization and merging over multiple machines in a commercial data center
- An experimental demonstrator for quantitative and qualitative evaluation of the proposed methods

The remainder of this paper is organized as follows: We first give an overview of the system architecture in Sec. 3.2. Onboard algorithms are presented in Sec. 3.3. After presenting the data representation and communication protocol in Sec. 3.4 we introduce the optimization and merging algorithms in Sec. 3.5. Finally, the evaluation results of our implementation are presented in Sec. 3.6 and we conclude in Sec. 4.5.

3.2 System Architecture

Real-time constrains, data I/O, network bandwidth, and computational requirements played an important role in the design choices of the proposed architecture. Generally,

processes that were sensitive to network delays or which connected high-bandwidth sensors were run on the robot, while computation- or memory-intensive processes without hard realtime constraints were run in the cloud.

Our architecture, see Fig. 3.2, mainly consists of

- *mobile robot*: low-cost robots, each with an RGB-D sensor, smartphone-class processor and a wireless connection to the data center, see Fig. 3.3.
- *robot clone*: A set of processes for each robot connected to the cloud that manages key-frames and other data accumulation tasks, while updating the robot with optimized (or post-processed) maps. Currently, the *robot clone* sends the pre-programmed motion commands to the *robot*. This 'cloud-based control' functionality can be extended in the future to do motion planning based on the map being built, see Fig. 3.2.
- *database*: a database for storing maps. A relational (MySQL) database and a non-relational database (MongoDB) was used for comparison.
- *map optimizer*: Parallel optimization algorithm to find the optimal pose graph based on all accumulated key-frames. After each optimization cycle, the *map optimizer* updates the database and triggers the *robot clone* to update the robot with the new map.
- *map merger*: This process tries to match frames from different maps. Once a match is found, transformations between two maps are computed and the two maps are merged into a single map.

All computational processes run on Rapyuta [2], a cloud Robotic platform that manages the computational processes and handles the robots' bidirectional communication and authentication. See Sec. 3.2 for more details.

Robot

Our robots, shown in Fig. 3.3, consist mainly of off-the-shelf components. The differential drive base of the iRobot Create provides the serial interface for sending control commands and receiving sensor information. PrimeSense CARMIN 1.08 is used for the RGB-D sensing, and provides two registered depth and color images in VGA resolution at 30 frames per second. A 48×52 mm embedded board with a smartphone-class multi-core ARM processor is used for onboard computation. The embedded board runs a standard Linux operating system and connects to the cloud through a dual-band USB wireless device. In addition to running the RGB-D sensor driver and controlling the robot, the onboard processor also runs a dense visual odometry algorithm to estimate the current pose of the robot. The key-frames produced by the visual odometry are sent to the cloud processes through the wireless device. See Sec. 3.3 for more information on the visual odometry algorithm. For a voxel-based approach see [19].

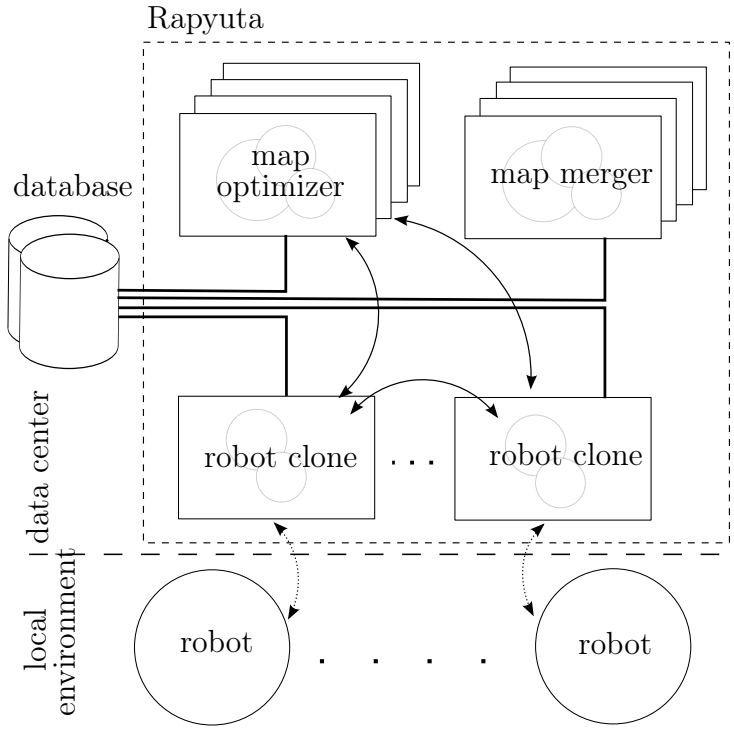


Figure 3.2: The overview of the proposed architecture based on Rapyuta: Each robot has a corresponding clone in the cloud. The clone is a set of processes (light-gray circles) running under a secured computational environment (rectangular boxes). Every computational environment has its own ROS master (dashed circles) and Rapyuta acts as a multi-master connecting processes running in different environments. Map optimization and merging are parallelized using multiple computational environments (stacked rectangles). All processes running inside the computational environments have a high bandwidth access to the database (cylinders). Robots have a WebSocket-based bidirectional full duplex connection (dotted curved lines) to their corresponding clones in the cloud.

The Cloud and Software

We use Rapyuta [2], a cloud robotics platform we developed previously, to run all our processes in the cloud. Since Rapyuta uses the WebSocket protocol to communicate with the robots, the robots and mapping processes need not be in the same network as they were in [4] and [11]. This allows us to seamlessly connect our robots in Zurich, Switzerland to a commercial Amazon [20] data center in Ireland. Furthermore, since WebSockets allow for persistent connection between processes, the processes running in the cloud can push data/updates to the robots without the robots having to periodically poll for updates.

Rapyuta can spawn multiple secure ROS-compatible computing environments, launch processes inside these computing environments, and facilitate the communication between these processes (even across different computing environments). This allowed graceful scaling of *map optimizer* and *map merger* processes in experiments. Moreover, Rapyuta enables custom message converters to be employed between the robot and the cloud. This flexibility enabled us to use optimal compression schemes, compared to the standard ROS

Figure 3.3: The two low-cost (< 600$) robots used in our evaluations: Each robot consists mainly of a differential drive base (iRobot Create), an RGB-D sensor (PrimeSense), an ARM-based single board computer (ODROID-U2), and a dual band USB wireless device.

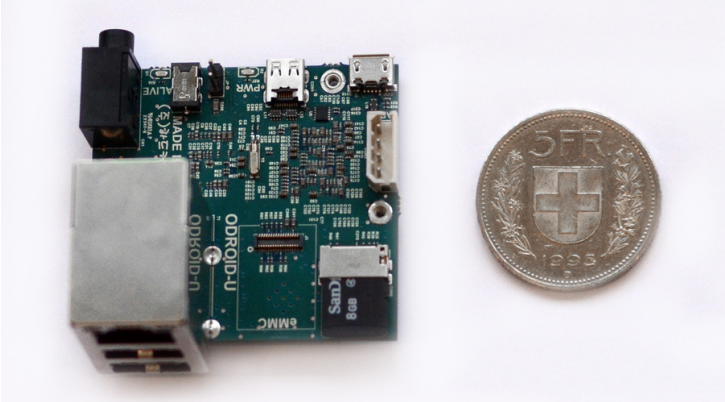


Figure 3.4: The onboard processor Odroid-U2: 48 × 52 mm embedded board with a smartphone-class quad core ARM Cortex-A9 processor.

message used in [11]. Visit `http://rapyuta.org/` for more details on Rapyuta.

3.3 Onboard Visual Odometry

In order to build a map of the environment it is necessary to track the position of the robot over time. Although several methods (such as wheel odometry, visual odometry, and the use of Inertial Measurement Units) provide information on the relative motion of the robot, only a few of these (i.e. visual odometry) provide the option to remove the accumulated errors with global optimization. The dense visual odometry algorithm used

on board the robots is largely inspired by [21], [22], and [23].

Preliminaries

This subsection defines some concepts and introduces the symbols used throughout the paper. Let

$$\begin{aligned}\mathcal{I} &:\mathbb{R}^2 \to [0,1]^3,\\ \mathcal{Z} &:\mathbb{R}^2 \to \mathbb{R}_+,\end{aligned}$$

represent the (color) intensity image and depth image of the camera respectively. To represent the camera's rigid body motion we use the twist vector $\xi \in \mathbb{R}^6$ and define $\hat{\xi}$ as

$$\hat{\xi} := \begin{pmatrix} 0 & -\xi(3) & \xi(2) & \xi(4) \\ \xi(3) & 0 & -\xi(1) & \xi(5) \\ -\xi(2) & \xi(1) & 0 & \xi(6) \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The over parametrized transformation matrix T can now be expressed as

$$T = \begin{pmatrix} R \in SO(3) & t \in \mathbb{R}^3 \\ 0^{1\times 3} & 1 \end{pmatrix} = \exp(\hat{\xi}).$$

Using a pinhole camera model, the projection π, and the inverse projection π^{-1} between the 3D point $\mathbf{p} := (X, Y, Z)$ and its corresponding pixel representation, $\mathbf{x} = (x, y)$ is given by

$$\begin{aligned}\mathbf{x} = \pi(\mathbf{p}) &= \left(\frac{X f_x}{Z} + o_x, \frac{Y f_y}{Z} + o_y\right),\\ \mathbf{p} = \pi^{-1}(\mathbf{x}, Z) &= \left(\frac{x - o_x}{f_x} Z, \frac{y - o_y}{f_y} Z, Z\right),\end{aligned}$$

where f_x, f_y denotes the focal lengths and o_x, o_y denotes the image center. Note that the second argument of inverse projection for our scenario comes from the corresponding depth pixel $\mathcal{Z}(\mathbf{x})$.

Given a frame, a tuple consisting of $\mathcal{I}$, $\mathcal{Z}$ and some other information (See Sec. 3.4), the warp of its pixel $\mathbf{x}$ to a frame with the relative pose M is given by

$$w(\mathbf{x}, M) := \pi\left(M\pi^{-1}(\mathbf{x}, \mathcal{Z}(\mathbf{x}))\right).$$

Finally, Key-frames are a subset of frames that in a way summarizes the full set. The

key-frames are also used as a base/reference to represent the pose of other frames.

Dense Visual Odometry Algorithm

The dense visual odometry algorithm starts with an empty set of key-frames. When it receives the first pair of color and depth images, they are added to the map as an initial key-frame with the initial pose. A map in our scenario consists of key-frames and their corresponding poses.

After initialization, the dense visual odometry algorithm estimates the pose of the camera based on each incoming frame from the camera. This pose estimation is done by minimizing the photometric error between the intensity images of the current frame and the key-frame given by

$$\begin{aligned} R_k(M_k) &= \sum_{\mathbf{x}\in\Omega} (\mathcal{I}(\mathbf{x}) - \mathcal{I}_k(w(\mathbf{x}, M_k)))^2, \\ &=: \sum_{\mathbf{x}\in\Omega} r_{\mathbf{x}}^2(M_k), \end{aligned} \tag{3.1}$$

where Ω is a set of all pixels that are available in both frames that were not occluded while warped, and M_k is the relative pose of the key-frame with respect to the current frame. The key-frame that is closest to the last estimate of camera pose is used as the current key-frame.

To minimize the non-linear cost function given in (3.1) with respect to M_k we use the Gauss-Newton method for solving non-linear least-squares problems [24]. Here, the j^{th} iteration is given by

$$\begin{aligned} M_{j+1} &= \exp(\bar{\xi}) M_j, \\ \bar{\xi} &= -(J_j^{\mathrm{T}} J_j)^{-1} J_j^{\mathrm{T}} r(M_j), \end{aligned}$$

where J_j is the Jacobian of the residual

$$r(M_j) := [r_{\mathbf{x}}(M_j)]_{\mathbf{x}\in\Omega}$$

and M_j is initialized with

$$M_0 = \begin{bmatrix} I_{3\times 3} & 0_{3\times 1} \\ 0_{1\times 3} & 1 \end{bmatrix}.$$

This iteration converges to

$$\arg\min_{M_k} R_k(M_k) = \lim_{j\to\infty} M_j.$$

At every iteration the Jacobian J_j can be calculated using the following chain rule

$$J_j = - \frac{\partial \mathcal{I}_k(w(\cdot))}{\partial w} \cdot \frac{\partial w(\cdot, \exp(\hat{\xi}) M_j)}{\partial \exp(\hat{\xi})} \cdot \left. \frac{\partial \exp \hat{\xi}}{\partial \xi} \right|_{\xi=0}.$$

Note that the first term in the right-hand side is the color gradient and the other terms can be calculated analytically. The implementation of this algorithm was optimized to run on a multi-core ARM processor. All operations, such as color conversion, sub-sampling, image gradient computation, and 3D point re-projection are parallelized. These operations involve independent per pixel operations, so they can be easily parallelized by splitting all pixels between several CPU cores. To achieve this we use a Threading Building Blocks library [25], which provides templates for easy code parallelization. In particular the *parallel_for* and *parallel_reduce* templates are used heavily in our implementation. We also use the auto-vectorization tools of the GCC compiler, which automatically replaces the regular instructions with specialized vector instructions where possible.

Since 3D point positions and image gradients are needed only for key-frames, they are computed only when a new key-frame is added (0-2 FPS depending on the robot speed). All images are stored in fixed-point values format (8-bit for intensity images and 16-bit for depth images), which may decrease the accuracy due to the rounding errors, but significantly improves the computational efficiency compared to processing images represented with floating-point values. With our implementation we were able to achieve a processing time of 15-20 [ms] for QVGA depth and color images with all 4 cores of CPU where loaded at approximately 60%. During this process, the visual odometry algorithm adds a new key-frame when the distance or the angle to the nearest key-frame in the map exceeds a predefined threshold.

3.4 Map Representation and Communication Protocol

Every map is a set of key-frames and a key-frame is a tuple represented as

$$(k, \mathcal{I}_k, \mathcal{Z}_k, q_k, t_k, \mathbb{I}_k),$$

where k is a global index of the key-frame, $\mathbb{I}_k$ is the intrinsic parameters of the camera, q_k the unit quaternion and t_k the translation vector. Note that q_k and t_k together represent the pose of the key-frame in the coordinate system of the current robot map. In the current implementation the global index k is a 64-bit integer, where the first 32-bits are used to identify the robot and the rest are used to index the key-frames collected by that robot. This indexing scheme saves approximately 4 billion key-frames from 4 billion robots, which is far beyond current needs.

The map is synchronized using the protocol shown in Fig. 3.5. When the visual odome-

try adds a new key-frame to the local map, it also sends one to the *robot clone*. All depth and color images are compressed with PNG for transmission. PNG is a lossless image compression that supports RGB and gray-scale images with up to 16-bit per pixel.

Once the key-frame has reached the *robot clone*, it is added to the database; the *map optimizer* process includes this key-frame in its next cycle. The *map optimizer* triggers the *robot clones* after the end of each cycle in order to update the local map on the robot. Once triggered, the *robot clone* gets the key-frame IDs of the local map on the robot, retrieves the updated key-frame pose from the database, and sends it back to the robot. The bandwidth requirement of this map update protocol is relatively low, since the update does not include any images/key-frame transmissions.

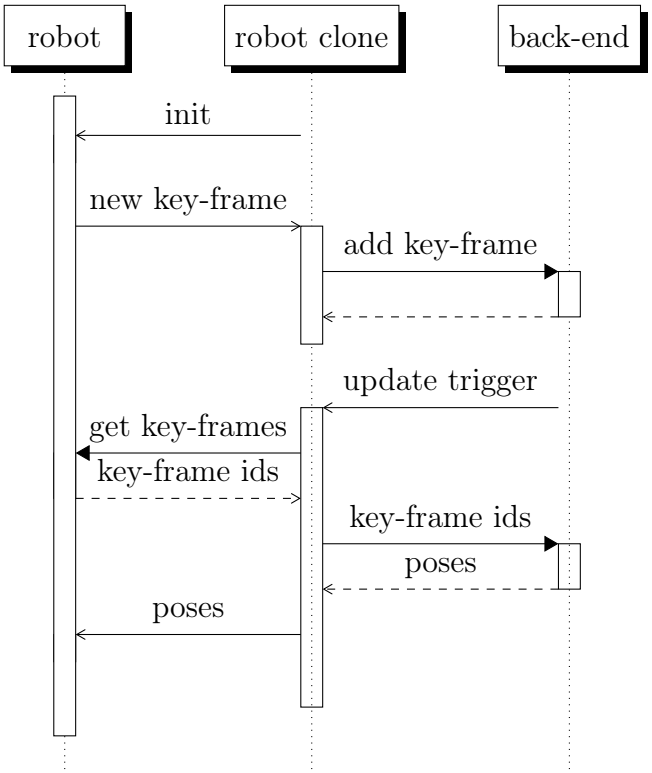


Figure 3.5: Sequence diagram of the proposed map synchronization protocol: All key-frames from the robot are sent to the robot clone, processed and saved to the back-end database. After every cycle of back-end optimization, the *robot clone* gets the local key-frame IDs from the *robot* and updates the local key-frame poses.

3.5 Map Optimization and Merging

The visual odometry that runs on the robot accumulates errors over time and causes a drift in the key-frame pose. This section presents the optimization techniques used to reduce the accumulated errors; these techniques work by minimizing error measures that include all acquired key-frames.

Map Initialization

Although this map initialization step is optional, it is recommended since it allows for the calibration of the camera intrinsic parameters. Further, where map initialization was used in experiments, the highly optimized initial map resulted in increased tracking stability.

During initialization the robot makes a 360 [°] in-place rotation. Assuming pure rotation allows to use well-established methods such as panorama optimization to be used. Our map-initialization is based on [26] and it globally optimizes all key-frame poses and the intrinsic camera parameters. When pure rotation is assumed, pixels from k'-th key-frame can be transformed to k-th key-frame by simply multiplying with the homography matrix

$$H_{kk'} = K R_k^{-1} R_{k'} K^{-1},$$

where R_k and $R_{k'}$ are rotation matrices of the key-frames k and k' with respect to a common reference, and K is an intrinsic calibration matrix parametrized by f_x, f_y, o_x, and o_y (see Sec. 3.3). In order to find the optimal key-frame orientations and the intrinsic parameters, one must find the parameter vector

$$p = (f_x, f_y, o_x, o_y, R_0, ...R_N),$$

that minimizes the per-pixel error of each overlapping pair of frames k, k' given by

$$E(p) = \sum_{k,k'} \sum_{\mathbf{x} \in \Omega_i} (\mathcal{I}_k(\mathbf{x}) - \mathcal{I}_{k'}(H(p)\mathbf{x}))^2$$

where $\mathcal{I}_k$ and $\mathcal{I}_{k'}$ are intensity images of the overlapping key-frames k and k'. The minimization of $E(p)$ with respect to p was performed using the Gauss-Newton method after parametrizing the updates using the Lie Algebra (see [26] for details). After the optimization, the floor of the map is selected using RANSAC and the XY-plane of the (world) coordinate frame was aligned with the floor.

Map Optimization

The global optimization of a map reduces errors accumulated during visual odometry, and consists of two main steps:

- Step 1: Construct a graph $\mathcal{G}$ where: 1) every key-frame of the map has a corresponding node; and 2) an edge between two nodes exists if the corresponding key-frames overlap and a relative transformation can be determined from the image data.
- Step 2: Solve the graph-based non-linear least squares problem given by:

$$\mathbf{p}^* = \arg\max_{\mathbf{p}} \sum_{i,j \in \mathcal{G}} \| \, e(p_i, p_j, p_{ij}) \, \|_2^2,$$

where $\mathbf{p} = [p_i]_{i \in \mathcal{G}}$ is the pose vector of all key-frames, p_{ij} is the constraint due to the overlap of key-frames i and j (calculated in Step 1), and e is an appropriate error measure that describes how well the pair p_i, p_j satisfy the constraint p_{ij}. In our case we are using the error function that minimizes the translational error and the rotational error (magnitude of the real part of the unit quaternion that represents the rotational error) both equally weighted.

Once the graph is constructed, several state-of-the-art open source frameworks such as g2o [27] and Ceres [28] can be used to solve Step 2. Our architecture uses g2o for step 2. Since construction of the graph $\mathcal{G}$ in Step 1 involves the matching of key-frames, which is a computationally expensive task, we parallelize this process over multiple machines as shown in Fig. 3.6.

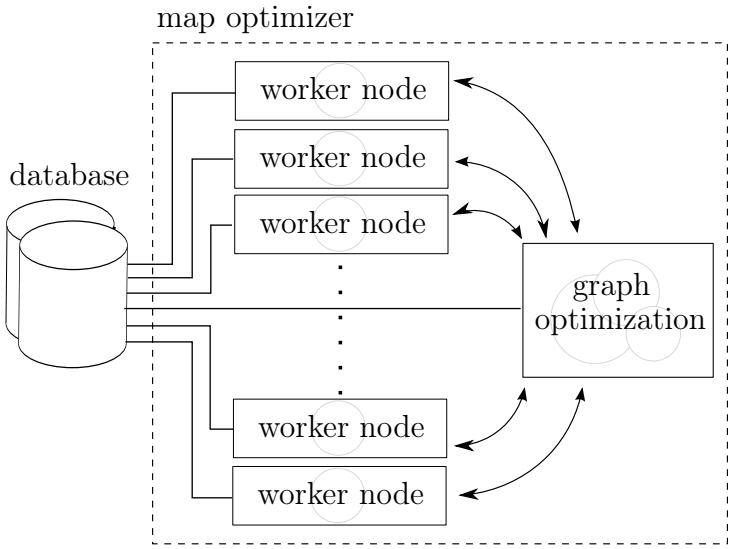


Figure 3.6: Map optimization architecture: Pose graph construction is distributed among worker nodes and the constructed graph is optimized in the graph optimization node.

The *graph optimization* node retrieves pairs of key-frame indexes from the database, which don't have a transformation yet, and distributes these between *worker nodes*. Note that the *graph optimization* node only selects the key-frame pairs that are within a distance threshold in order to limit the exponential increase of the number of key-frame pairs.

The *worker nodes* try to compute the transformation between each pair of key-frames they receive. To compute the transformation, *worker node* loads the precomputed SURF keypoints for these key-frames and their respective 3D positions from the database and tries to find a transformation between them using RANSAC. If it succeeds, it saves the transformation to the database and proceeds to the next pair. Once all *worker nodes* have finished, the optimization node optimizes the error function, completing the optimization cycle. After every optimization cycle, key-frame poses are updated in the database and an update trigger is sent to the *robot clones* to update the local map on the robot. The

graph structure of each map is stored as a table of pairs in a database and updated every time the new key-frames are added.

Map Merging

During collaborative mapping the robots can enter areas that have already been already explored by other robots. Being aware of the overlaps significantly decreases the mapping time and increases the map accuracy.

For the collaborative mapping, no prior knowledge on the initial robot poses is assumed and robots starts out with a separate map. The map merging runs as background process, continuously selecting a random key-frame from a map in the database and trying to find a matching key-frame from the other map. The process extracts SURF key-points from these key-frames and tries to match them using RANSAC. If a valid transformation is found, all key-frame poses of the smaller map are transformed into to the coordinate system of the other and the database entries are updated with the new values. Note that except for a minor difference in database update logic, the same *worker nodes* of the map optimization can be reused to parallelize map merging. Figure 3.1 shows a map merged from two robots and the re-localized robots in the new map.

3.6 Evaluation

The experimental setup for evaluation consisted of two low-cost robots (Fig 3.3) in Zurich, Switzerland and the cloud-based architecture (Fig. 3.2) running in Amazon's data center in Ireland. In addition to qualitatively evaluating the building and merging of maps created in different environments as shown in Figs 3.1 (72 key-frames) and 3.7 (423 key-frames), we quantitatively evaluated network usage, localization accuracy, and global map optimization times.

Figures 3.8 and 3.9 show the network usage of the robot executing a 360 [°] in-place rotation and a 2 [m] straight motion with different speeds. It is clearly visible that bandwidth is proportional to the velocity of the robot, with the highest bandwidth about 500 [KB/s]. For purposes of comparison, note that our cloud-based KinectFusion [29], a dense mapping algorithm, uses around 3.5 [MB/s] since all frames must be sent to the cloud for processing. For more details on this demonstrator and the video compressions used visit `https://github.com/IDSCETHZurich/rapyuta-kinfu`.

To evaluate the accuracy of visual odometry and influence of the global optimization, a high precision commercial motion tracking system was used. Figures 3.10 and 3.11 show the translation and rotation(yaw) errors error of the visual odometry with and without the cloud-based global optimization during a 360 [°] in-place rotation. Figure 3.12 shows translation error for a 2 [m] straight line motion. The yaw error during the straight line motion was below 0.01 [rad] for both the optimized and the non-optimized visual odometry. In all cases (Figs.3.10-3.12) the cloud-based optimization was able to reduce

the errors significantly, especially when there is a loop closure, such as in Figs. 3.10 and 3.11. Note that, due to the relatively low visual features in the motion capture space, the maps of this space were of low quality compared to the ones given in Figs 3.1 and 3.7.

Finally, Fig. 3.13 shows the time taken for map optimization against the number of *worker nodes*. Although the processing time initially decreases with the number of *worker nodes*, this decrease later vanishes due to communication latencies. The measurements also show that the gain due to parallelization is significantly more for larger sets of key-frames. To reduce latencies due to database access during map optimization, we compared a relational and a non-relational database with respect to their I/O speeds. MySQL was used to represent relational databases, whereas MongoDB was used to represent non-relational databases and the results are shown in Figs. 3.13a and 3.13b. Although both databases gave a similar performance with respect to speed, using the `JOIN` clause of MySQL (join clause combines records from two or more tables in a database), a significant amount of computation was offloaded from the *graph optimization* node to the database during the key-frame pair selection (see Sec. 3.5).

3.7 Conclusion

We presented first steps towards a scalable cloud robotics service for mapping and localization using Rapyuta [2], an open-source cloud robotics framework we developed in our previous work.

First, we provided an open source implementation of a state-of-the art, dense visual odometry algorithm on a smartphone-class ARM multi-core CPU. Second, we developed a data protocol that sends only compressed key-frames to the cloud, reducing the bandwidth requirements . In addition, the protocol allows the cloud processes to push key-frame pose updates back to the robots without the need for constant polling. Third, we presented techniques for parallelizing the computationally expensive operations of map optimization and map merging in a commercial data center, and provided a corresponding open source software implementation[1].

As illustrated by our demonstrator, this cloud-based architecture holds the potential to greatly increase the number of mobile robot platforms capable of creating large, high-quality maps and performing accurate localization. The robots used were entirely built using low-cost, off-the-shelf components, i.e., an Odroid-U2 board (USD 90), a PrimeSense CARMIN RGB-D sensor (USD 200), a simple iRobot Create robot base (USD 220), and a USB wireless device (USD 40)). Further, the commercial cloud-infrastructure pro-

`http://github.com/IDSCETHZurich/rapyuta-mapping`

Note that the bandwidth depends on many factors such as the speed of the robot and thresholds used for generating key-frames. Therefore it is hard to do a rigorous comparison with other works such as [11]. However, our advantage compared to [11] include scalability, the non-requirement for the robot and the server to be in the same network, and our PNG-based custom converters to achieve a better compression compared to the ROS standard messages.

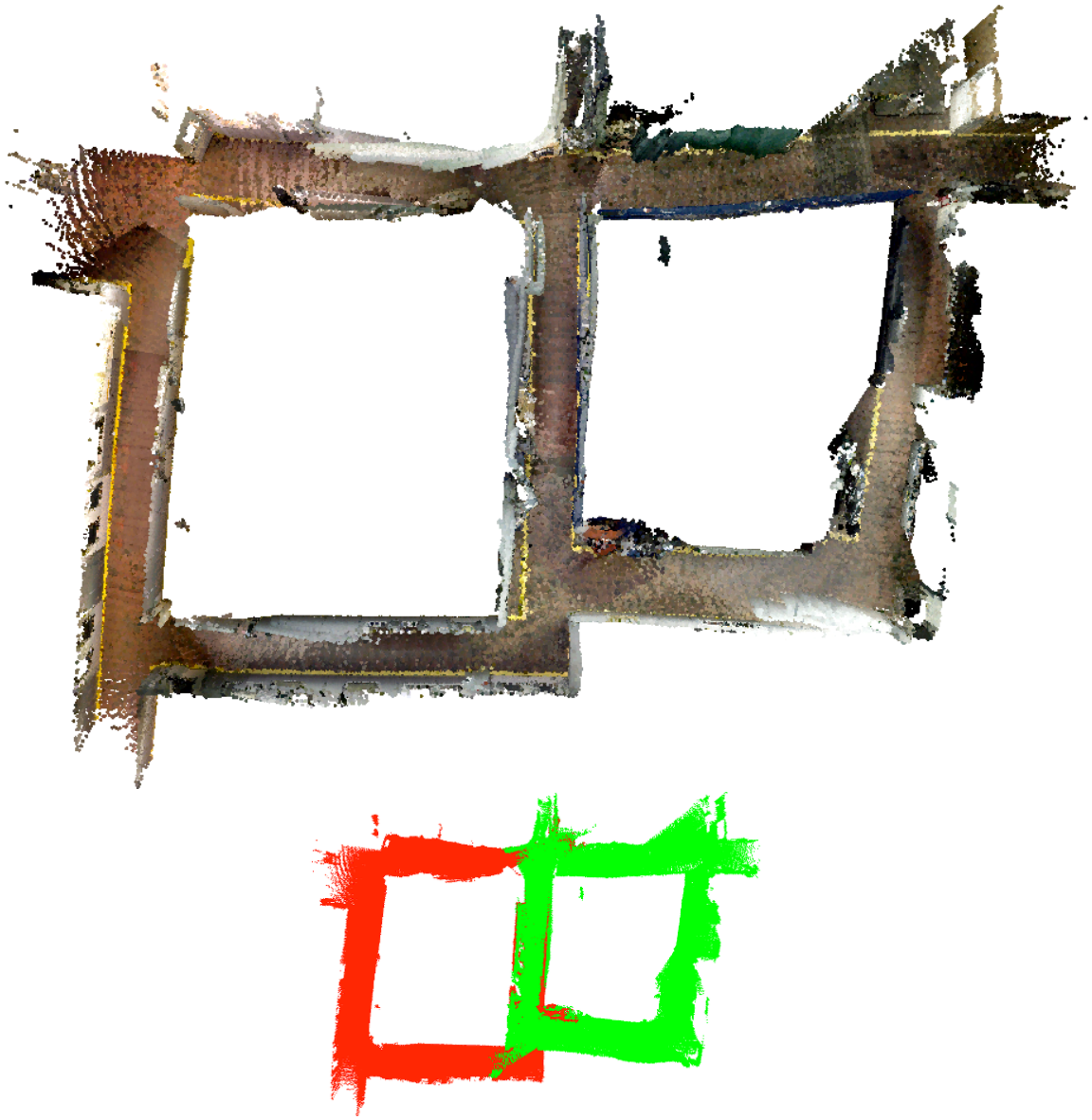


Figure 3.7: A point cloud map of a 40m-long corridor. The map was collaboratively built by two robots, and consists of 423 key-frames. Different colors in the right image show parts of the map built by different robots.

vides computational services at very low cost (USD 0.130 per hour for every *m1.medium* instance (∼2×1.7 GHz, 3.75 GB) [20].

Finally, we showed both qualitative and first quantitative results achieved with the architecture. As shown in Figs. 3.1 and 3.7 as well as in the accompanying video, our implementation yields maps comparable to those obtained with more expensive robot hardware. First quantitative experiments confirmed that bandwidth requirements are well within those typically available in modern wireless networks (< 0.5 [MB/s]). They also confirmed that map optimization provided via the cloud significantly reduces uncertainty of the robot's visual odometry. Moreover, they confirmed the computational advantage

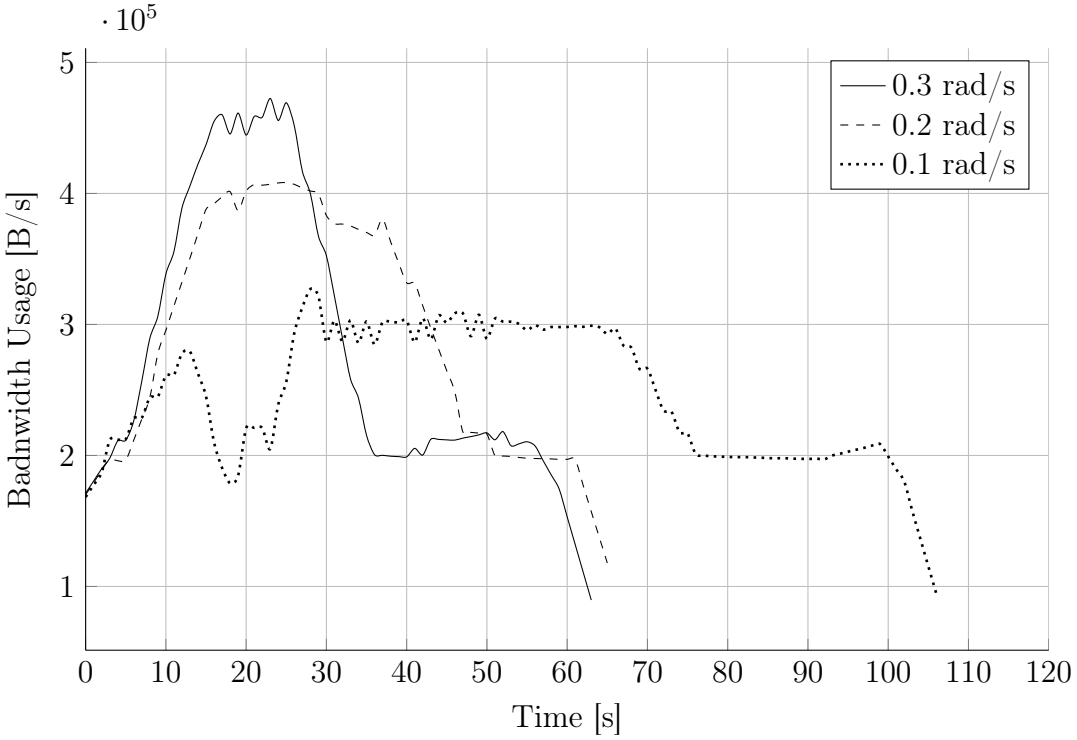


Figure 3.8: Network usage in bytes per second for a single robot performing a 360 [°] in-place rotation.

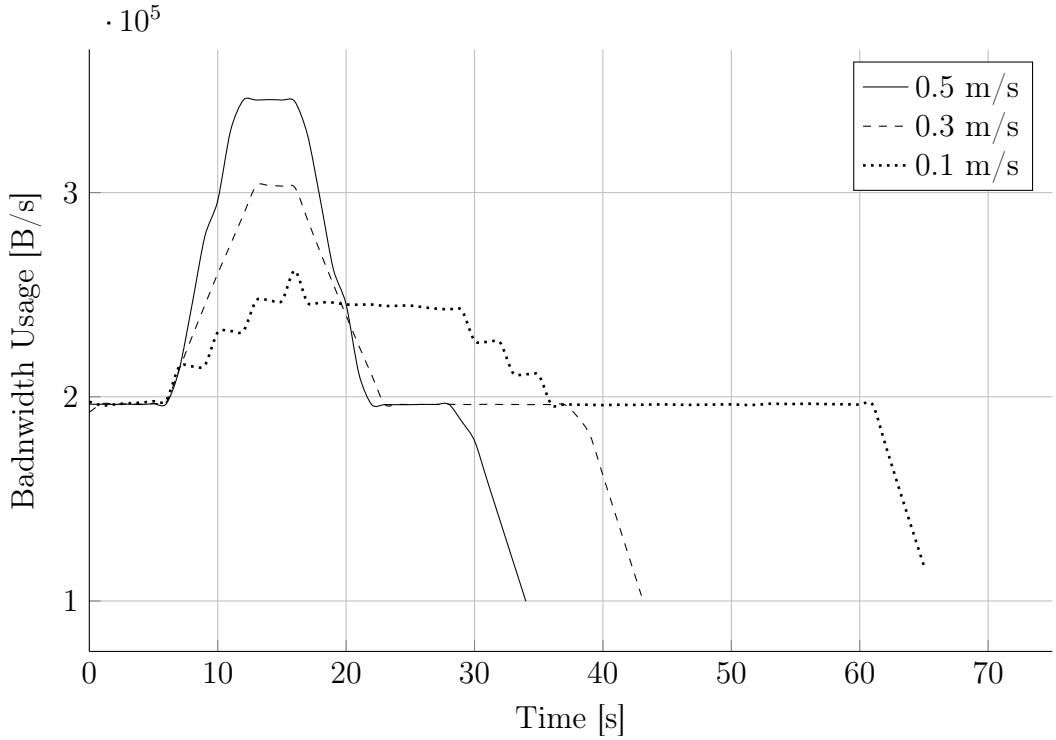


Figure 3.9: Network usage in bytes per second for a single robot performing a 2 [m] straight motion.

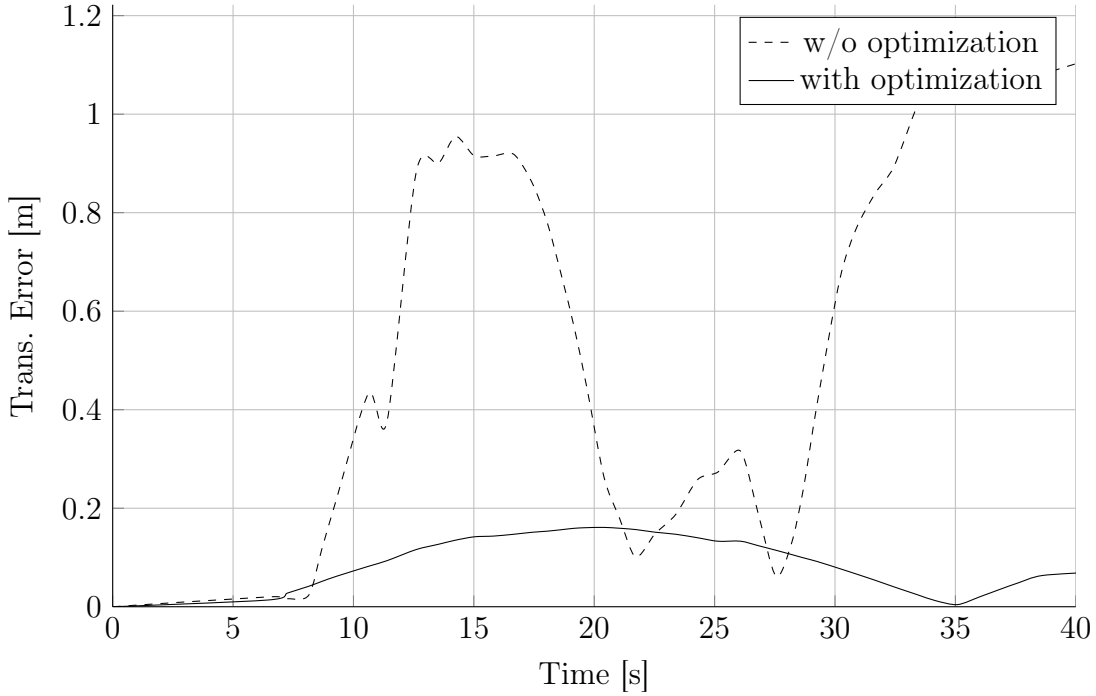


Figure 3.10: Translation error of key-frames extracted by visual odometry during a 360 [°] in-place rotation with and without map optimization. The errors are based on the ground truth measurements from VICON, a high-precision motion capture system.

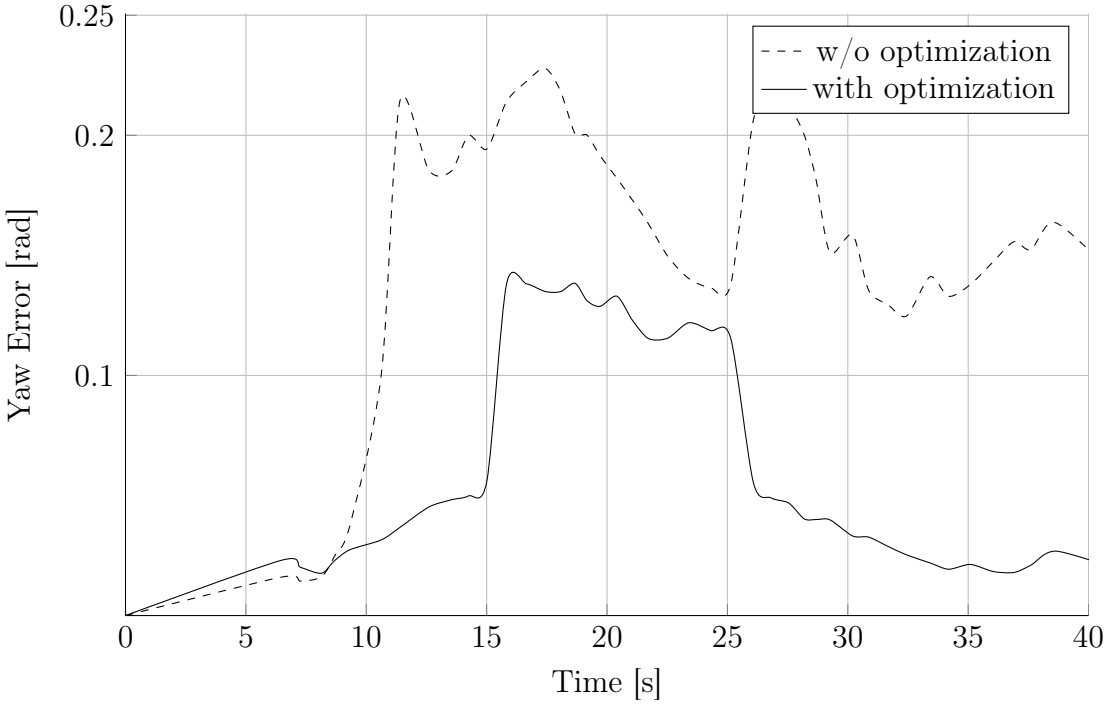


Figure 3.11: Rotation error of key-frames extracted by visual odometry during 360 [°] in-place rotation with and without map optimization. The errors are based on the ground truth measurements from VICON, a high-precision motion capture system.

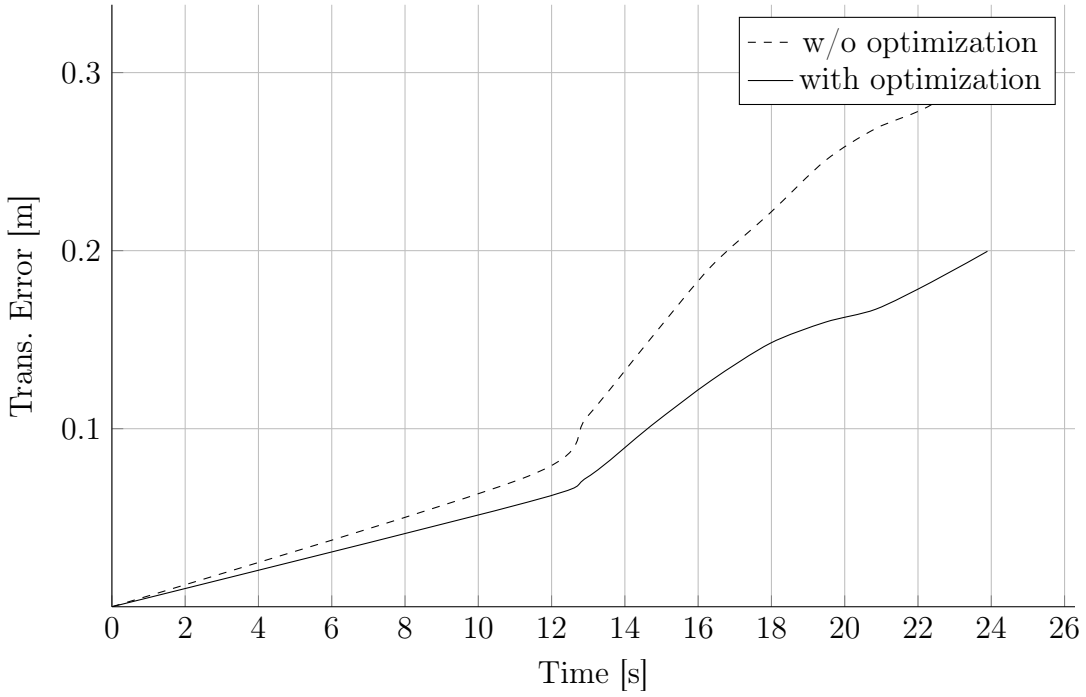


Figure 3.12: Translation error of key-frames extracted by visual odometry during a 2 [m] forward motion with and without map optimization. The errors are based on the ground truth measurements from VICON, a high-precision motion capture system.

of parallelization for map optimization in the cloud.

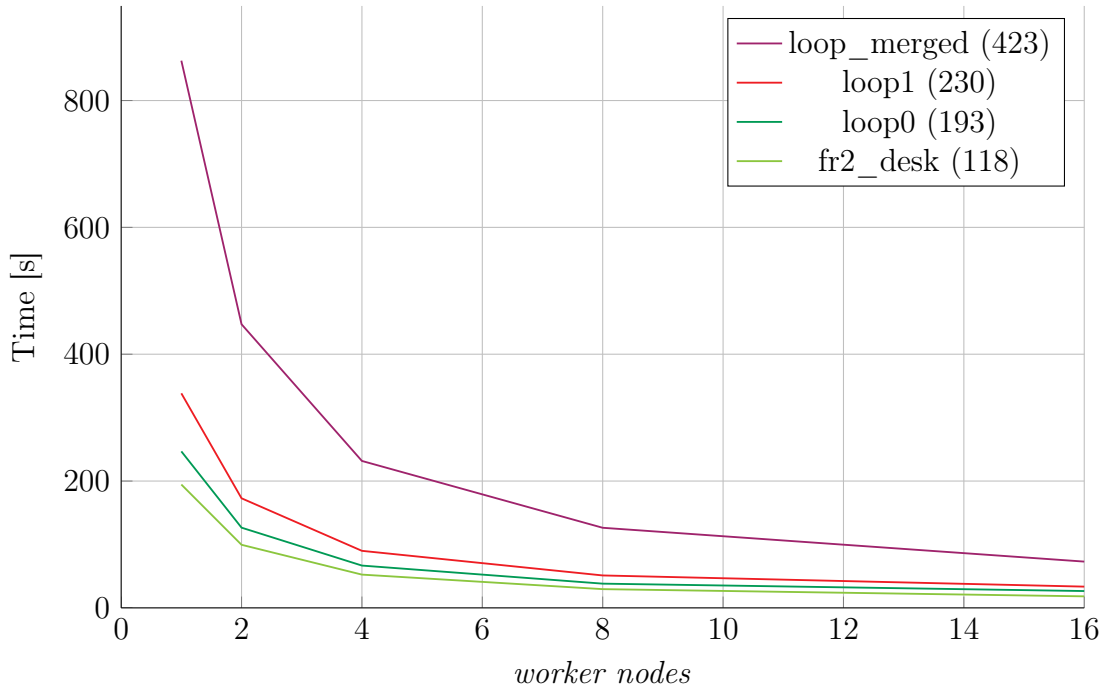
Possible future improvements include the incorporation of the depth error into visual odometry [22], substituting the current naive bag-of-words-based place recognition to a more probabilistic approach such as FAB-MAP [31] for map merging, and the creation of larger maps using more robots.

Acknowledgement

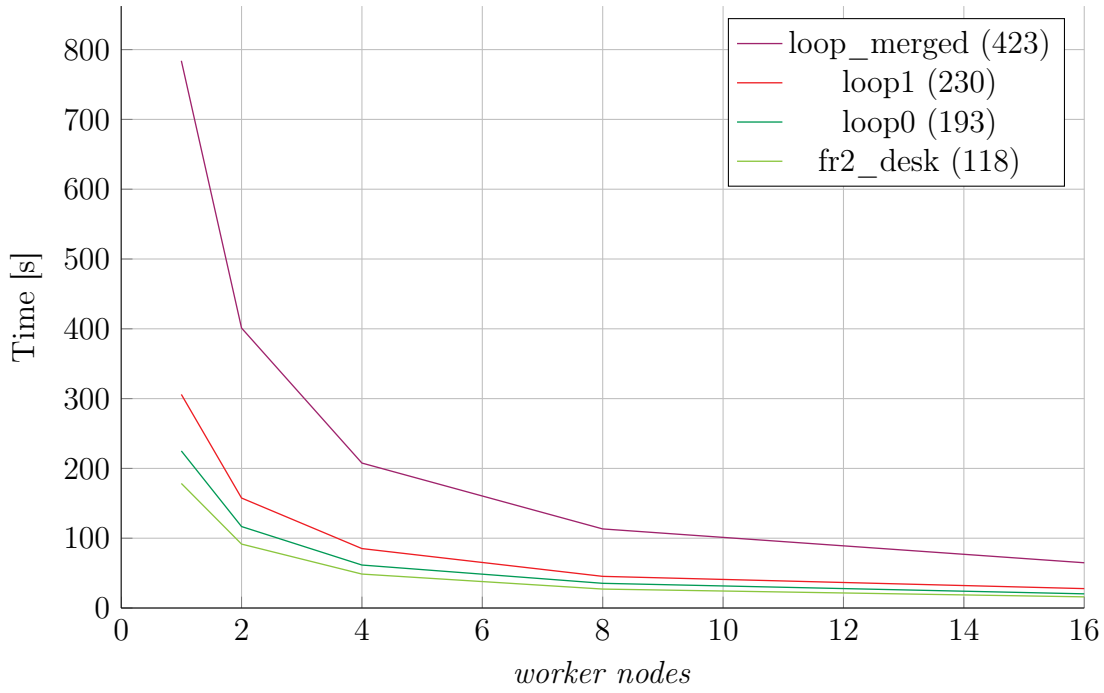
This research was funded by the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement no. 248942 RoboEarth. This work also received support from AWS (Amazon Web Services) in Education Grant award. The authors would like to thank Dejan Pangercic for his continuous support and motivation, and their colleagues Dominique Hunziker and Dhananjay Sathe for their support with Rapyuta.

References

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Special Publication 800-145, 2014, available http:



(a) MySQL



(b) MongoDB

Figure 3.13: Map optimization times against the number of *worker nodes*. The numbers in parenthesis in the legend denote the number of key-frames. loop0 and loop1 are the two loops of the corridor shown in Fig. 3.7. loop_merged is a combination of both. fr2_desk is a public data set obtained from [30].

//csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[2] G. Mohanarajah, D. Hunziker, M. Waibel, and R. D'Andrea, "Rapyuta: A cloud robotics platform," *IEEE Trans. Automation Science and Engineering*, February 2014, accepted.

[3] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft, "RoboEarth," *Robotics Automation Mag., IEEE*, vol. 18, no. 2, pp. 69–82, June 2011.

[4] R. Arumugam, V. R. Enti, B. Liu, X. Wu, K. Baskaran, F. K. Foong, A. S. Kumar, D. M. Kang, and W. K. Goh, "Davinci: A cloud computing framework for service robots." in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, May 2010, pp. 3084–3089.

[5] K. Kamei, S. Nishio, N. Hagita, and M. Sato, "Cloud Networked Robotics," *IEEE Network*, vol. 26, no. 3, pp. 28–34, May-June 2012.

[6] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, "Cloud-based robot grasping with the google object recognition engine," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2013, pp. 4263–4270.

[7] B. Kehoe, D. Berenson, and K. Goldberg, "Toward cloud-based grasping with uncertainty in shape: Estimating lower bounds on achieving force closure with zero-slip push grasps," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2012, pp. 576–583.

[8] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[9] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE Sym. Mass Storage Systems and Technologies (MSST)*. IEEE, May 2010, pp. 1–10.

[10] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, May 2009.

[11] L. Riazuelo, J. Civera, and J. M. M. Montiel, "C2TAM: A cloud framework for cooperative tracking and mapping," *Robotics and Autonomous Systems*, 2013, accepted for publication.

[12] G. Klein and D. Murray, "Parallel tracking and mapping on a camera phone," in *Proc. IEEE Int. Symp. Mixed and Augmented Reality*, October 2009, pp. 83–86.

[13] R. Goransson, A. Aydemir, and P. Jensfelt, "Kinect@Home: Crowdsourced RGB-D data," in *IROS Workshop on Cloud Robotics*, 2013. [Online]. Available: http://www.kinectathome.com/

[14] "Matterport," 2013. [Online]. Available: http://matterport.com/

[15] A. Howard, S. Gaurav S., and M. Maja J., "Multi-robot mapping using manifold representations," in *Proc. IEEE Int. Conf. Robotics Automation (ICRA)*, vol. 4, May 2004, pp. 4198—-4203.

[16] D. Zou and P. Tan, "Coslam: Collaborative visual slam in dynamic environments," *IEEE Tran. Pattern Analysis and Machine Intelligence*, vol. 35, no. 2, pp. 354–366, 2013.

[17] J. A. Hesch, D. G. Kottas, S. L. Bowman, and S. I. Roumeliotis, "Camera-imu-based localization: Observability analysis and consistency improvement," *Int. J. Rob. Res.*, vol. 33, no. 1, pp. 182–201, Jan. 2014.

[18] Google Inc., "Project Tango," 2014. [Online]. Available: http://www.google.com/atap/projecttango/

[19] J. Chen, D. Bautembach, and S. Izadi, "Scalable real-time volumetric surface reconstruction," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 113:1–113:16, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2461912.2461940

[20] Amazon.com Inc., "Amazon Elastic Compute Cloud," 2013. [Online]. Available: http://aws.amazon.com/ec2/am

[21] F. Steinbrucker, J. Sturm, and D. Cremers, "Real-time visual odometry from dense RGB-D images," in *ICCV Computer Vision Workshop*, November 2011, pp. 719—-722.

[22] C. Kerl, J. Sturm, and D. Cremers, "Robust Odometry Estimation for RGB-D Cameras," in *Proc Int. Conf. Robotics and Automation (ICRA)*, May 2013, pp. 3748–3754.

[23] T. Tykkala, C. Audras, and A. I. Comport, "Direct iterative closest point for real-time visual odometry," in *ICCV Computer Vision Workshop*, November 2011, pp. 2050—-2056.

[24] K. Madsen, H. B. Nielsen, and O. Tingleff, "Methods for non-linear least squares problems (2nd ed.)," 2004. [Online]. Available: http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3215/pdf/imm3215.pdf

[25] J. Reinders, *Intel Threading Building Blocks*, 1st ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2007.

[26] S. Lovegrove and A. J. Davison, "Real-time spherical mosaicing using whole image alignment," in *Proc. 11th Euro. Conf. Computer Vision*, September 2010, pp. 73–86.

[27] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2011, pp. 3607–3613.

[28] S. Agarwal and K. Mierle, "Ceres solver," 2014. [Online]. Available: http://code.google.com/p/ceres-solver/

[29] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Proc. IEEE Int. Symp. Mixed and Augmented Reality (ISMAR)*, October 2011, pp. 127–136.

[30] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. Int. Conf. on Intelligent Robot Systems (IROS)*, Oct. 2012.

[31] M. Cummins and P. Newman, "FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance," *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 647–665, 2008. [Online]. Available: http://ijr.sagepub.com/cgi/content/abstract/27/6/647

4

Gaussian Process Optimization-based Learning for Trajectroy Tracking

4.1 Introduction

Control systems are designed to regulate the behavior of dynamical systems and make them follow a human-assigned task or trajectory. They base their regulation on a physical *nominal* model, often derived from first principles. However, whenever there is an uncertainty in the model, such as repeating disturbances not taken into account in a complex environment, the control system will need to learn from its past behavior and compensate for the inaccuracy of the nominal model. Machine Learning tools can be employed in an *Adaptive Control* fashion to guide this learning process.

For systems that work in a repetitive manner, such as robotic manipulators and chemical plants, Iterative Learning Control (ILC) algorithms are used to improve on the performance. In ILC, the feed-forward control signal is modified in each iteration to reduce the error or the deviation from the given task or reference trajectory. A good analogy is a basketball player shooting a free throw from a fixed position: during each shot the basketball player can observe the trajectory of the ball and alter the shooting motion in the next attempt [1].

The limitation with ILC is that it assumes a fixed task or trajectory. While this is a reasonable assumption to make in some repeated tasks, ILC is not *learning* in the proper sense: it fails to generalize over different tasks and cannot handle the cases when the trajectories are modified. In all such cases, the Iterative Learning Controller will need to

This paper is submitted to the *31st International Conference on Machine Learning*, January 2014.

start from scratch, throwing away useful data. Data from different tasks can be used to generalize, a matter of paramount importance in complex tasks.

We therefore look at the problem of *generalization* and show that a significant amount of knowledge can be transferred even between cases where the reference trajectories are not the same. A basketball player does not have to learn the free throw motion from scratch each time he finds himself in a slightly different position. We call these cases or reference trajectories *contexts*. Context can change in a given task and it is the responsibility of the autonomous agent or the learning controller to adapt to different contexts.

The motivation for transfer learning under different contexts comes mainly from the RoboEarth project [2], a robot specific knowledge repository where robots share and learn from each others' experience. The ability to transfer knowledge even under different contexts will significantly improve the performance of future robots. For example, consider a robot learning to pour tea into a cup and over time perfecting the motion. The learned pouring motion can be uploaded to a central database, marked with the hardware-specifics of the particular robot as well as the size and shape of the teapot as context. Another robot with slightly different hardware, holding a different teapot, can download the stored motion as a prior and adapt it to its particular context, thereby eliminating the need to learn the motion from scratch.

In this paper, we introduce a reinforcement-learning (RL) algorithm called *TGP* that learns to track trajectories in state space online. *TGP* stands for *(trajectory) tracking with Gaussian Processes*. Specifically, the proposed algorithm uses Gaussian Process Optimization (GPO) in the bandit setting to track a given trajectory. It implicitly learns the dynamics of the system, and in addition to improving the tracking performance, it facilitates knowledge transfer between different trajectories.

Related Work

Gaussian Processes (GP) are increasingly applied in control theory, where they are used to learn the unknown system dynamics. In [3] the authors propose a hybrid-GP approach combined with reinforcement learning to control an autonomous blimp. Their framework requires them to learn the dynamics itself with multiple GP regressions whereas we use GPO to track the global minimum of an unknown, *scalar* cost function, as will be detailed in the next sections. In [4,5] the authors first learn the dynamics with GP regression. In the second step policies for a parameterized controller are learnt by propagating through the GP model. This algorithm, called PILCO, involves necessarily long offline calculations and does not have a guarantee for convergence. Unlike PILCO, *TGP* incorporates feedback and can generalize over different trajectories. Furthermore we prove that *TGP* converges to the tracked trajectory under mild assumptions.

Gaussian process optimization literature proposes several heuristics such as Expected Improvement [6] and Most Probable Improvement [7] for trading off exploration and exploitation. The first method with provable sublinear regret was proposed in [8] and extended to the contextual setting in [9]. We apply this approach to dynamical systems in this paper.

Summary

Our main contributions are as follows:

- We propose *TGP*, a reinforcement learning algorithm that efficiently learns to track a given trajectory.
- We prove that *TGP* under mild assumptions, learns to track the given trajectory arbitrarily well.
- With *TGP* we establish a novel connection with *transfer learning* and present a metric to quantify knowledge transfer.
- We show in numerical examples, the proposed approach and evaluate its performance both in trajectory tracking and in transfer learning by comparing with the state of the art control algorithms.

4.2 Problem Statement and Background

Consider the following discrete dynamics:

$$x_{t+1} = f(x_t, u_t), \quad t = 0, 1, \ldots, N, \tag{4.1}$$

where f is Lipschitz continuous in both arguments, $x_t \in \mathcal{X} \subset \mathbb{R}^n$ is the state of the system at time stage t and $u_t \in \mathcal{U} \subset \mathbb{R}^m$ is the control input.

In discrete time trajectory tracking, the control objective is to minimize the cost functional

$$J(\pi) = \sum_{t=1}^{N} d(x_t, s_t) \tag{4.2}$$

where the desired states s_t belong to a given trajectory $\Sigma = \{s_0, s_1, \ldots, s_N\}$ at each time stage, $\pi = \{u_0, \ldots, u_{N-1}\}$, and $d(x_t, s_t)$ is a suitably chosen semimetric, e.g. squared Euclidean distance. Note that this minimization should be done considering the state and input constraints and should satisfy (4.1). In general minimizing (4.2) is impractical and in Model Predictive Control (MPC), a very popular technique in control theory, the objective to minimize is instead $\sum_{t=1}^{M} d(x_t, s_t)$ with M typically much less than N, i.e. $M \ll N$.

However such receding horizon approaches may perform badly or suffer from instability when the dynamics f is not known very well. More precisely, consider the following dynamics typically derived from first principles:

$$\hat{x}_{t+1} = f_{\text{nom}}(x_t, u_t), \quad t = 0, 1, \ldots, N, \tag{4.3}$$

We discard here the penalties on the inputs u_t.

where f_{nom} is again Lipschitz continuous in both arguments. This *nominal* model may not be quite accurate when there are unmodelled dynamical effects or repeating disturbances not taken into consideration: (4.3) can fail to predict the future states. The foremost aim of this work is to develop a framework that can learn these unmodelled dynamical effects efficiently. To do this we learn to track at each time stage t the (global) minimum of an unknown scalar cost function $q_t = d(x_t, s_t)$.

The unknown cost function $q_t = d(x_t, s_t)$ at each time stage $t \geq 1$ depends on the previous state x_{t-1} and the desired state s_t as the *context*, the previously applied input u_{t-1} as the *action*. The contexts $c_t = (x_t, s_{t+1})$ for $t \geq 0$ in a more general setting can be imagined as revealed by an adversary from $\mathcal{C} = \mathcal{X}^2$, i.e. environmental variables independent of the input u_t to be applied. For brevity we sometimes use the notation $z = (u; c)$ for the variables in the joint input-context space $\mathcal{D} = \mathcal{U} \times \mathcal{C}$.

Regret.

A natural performance metric for an algorithm tracking the (global) minimum of a function $q(u; c)$ is the cumulative regret $R_T = \Sigma_{t=1}^{T} r_t$ or equivalently the cumulative reward (the negative of cumulative regret). Here $r_t = q(u_t; c_t) - q^*$ is the instantaneous regret where u_t is the action taken at time t, $q^* = \min_{u \in \mathcal{U}} q(u; c_t)$.

A desired feature of such an algorithm is to have asymptotically *no-regret*: $\lim_{T \to \infty} R_T/T = 0$ because then a subsequence of actions taken converge to the optimal action. Furthermore, any bounds on the average regret R_T/T after T rounds imply a convergence rate for this subsequence since the minimum $\min_{t \leq T} q(u_t; c_t)$ is sandwiched between q^* and the average.

Gaussian Processes and RKHS.

In order to guarantee no-regret, it is necessary to impose some sort of smoothness assumptions on the function $q(u; c)$. We can implicitly enforce smoothness on the function by assuming that $q(u; c)$ has low *complexity* as measured under an RKHS norm. *Reproducing kernel Hilbert spaces* [10] are intimately tied to GPs and their covariance functions $k(z, z')$. A *Gaussian Process* (GP) is a collection of dependent random variables, any finite number of which has a joint Gaussian distribution [11].

Gaussian processes, completely specified by a mean function $\mu(z)$ and a covariance function $k(z, z')$, can be seen as a prior distribution over the space of functions. For $y_i = q(z_i) + \epsilon_i$, $\epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$ with noisy observations $\mathbf{y}_T = \{y_1, \ldots, y_T\}$ at sample points $Z_T = \{z_1, \ldots, z_T\}$, the posterior over $q(z)$ is again a Gaussian Process distribution with mean $\mu_T(z)$ and covariance $k_T(z, z')$ given by the following simple analytic equations:

$$\mu_T(z_*) = \mu(z_*) + \mathbf{k}_T(z_*)^{\mathrm{T}}[\mathbf{K}_T + \sigma_n^2 \mathbf{I}]^{-1}(\mathbf{y}_T - \boldsymbol{\mu}_T) \tag{4.4}$$

$$k_T(z_*, z'_*) = k(z_*, z'_*) - \mathbf{k}_T(z_*)^{\mathrm{T}}[\mathbf{K}_T + \sigma_n^2 \mathbf{I}]^{-1}\mathbf{k}_T(z'_*) \tag{4.5}$$

$$\sigma_T^2(z_*) = k_T(z_*, z_*) \tag{4.6}$$

where $\boldsymbol{\mu}_T = [\mu(z_1), \ldots, \mu(z_T)]^{\mathrm{T}}$ is the prior mean at the sample points, $\mathbf{k}_T(z_*) = [k(z_1, z_*), \ldots, k(z_T, z_*)]^{\mathrm{T}}$ is the vector of covariances between the sample points and the test point z_* and $\mathbf{K}_T = [k(z, z')]_{z,z' \in Z_T} \succeq 0$.

With these update equations, Gaussian processes can be used in nonparametric regression to predict the mean and variance of unknown test points. Nonparametric regression methods have the advantage of avoiding rigidity encountered typically in regression with finite basis functions.

The RKHS $\mathcal{H}_k(\mathcal{D})$ corresponding to the GP covariance function $k(z, z')$ is a complete subspace of $L_2(\mathcal{D})$ with an inner product $\langle \cdot, \cdot \rangle_k$ obeying the reproducing property: $\langle q(\cdot), k(z, \cdot) \rangle_k = q(z)$ for all $q(u; c) \in \mathcal{H}_k(\mathcal{D})$. The induced RKHS norm $\|q(u; c)\|_k < \infty$ measures the smoothness of $q(u; c)$.

Bandits.

In the general reinforcement learning problem, contexts depend on the actions taken. In particular, in trajectory tracking, next states are determined by the previously applied input sequence. A simpler setting where the actions do not influence the contexts is known as *multi-armed bandits*, where in order to find the global minimum of an unknown noisy function, it must be *explored* by sampling at promising points u and must be *exploited* at the current minimum values. Using upper confidence bounds (UCB) is a particularly simple and intuitive heuristic to balance the exploration/exploitation tradeoff and works by keeping track of upper confidence bounds of the sample points [8].

In [9] the authors discuss such a learning algorithm which maintains a GP-estimate of the function to be optimized with (4.7):

$$u_t = \arg\min_{u \in U} \left(\mu_{t-1}(u; c) - \beta_t^{1/2} \sigma_{t-1}(u; c) \right) \tag{4.7}$$

where β_t is the parameter that balances exploration-exploitation at each time stage t. They show that the performance of this algorithm can be linked to the *maximal information gain* for the particular cost function to be learned:

$$\gamma_T = \max_{A \subset \mathcal{D}: |A| = T} \mathcal{I}(\mathbf{y}_A; q) \tag{4.8}$$

$$= \max_{A \subset \mathcal{D}: |A| = T} H(\mathbf{y}_A) - H(\mathbf{y}_A | q) \tag{4.9}$$

where $\mathcal{I}(\mathbf{y}_A; q)$ is the *information gain due to sampling* A. For a Gaussian, the entropy $H(\mathcal{N}(\boldsymbol{\mu}, \mathbf{K})) = \frac{1}{2} \log |2\pi e \mathbf{K}|$ using which we get: $\gamma_T = \frac{1}{2} \log |\mathbf{I} + \sigma_n^{-2} \mathbf{K}_A|$.

The cumulative regret of (4.7) can be bounded if $q(u; c) \in \mathcal{H}_k(\mathcal{D})$, i.e. $\|q(u; c)\|_k^2 \leq B$ for some $B < \infty$, where the kernel generating the RKHS is any valid combination of linear or squared exponential covariance function $k(z, z')$. The valid combinations of kernels

Matérn kernels with $\nu > 1$ can also be considered.

are summarized in [12]. In particular, tensor products and sums of linear and squared exponential kernels are allowed in our case. The bound on the cumulative regret is as follows:

$$R_T = \sum_{t=1}^{T} q(u_t; c_t) - q^* \leq \sqrt{\kappa T \beta_T \gamma_T} \tag{4.10}$$

with high probability $p \geq 1 - \delta$. Here $\kappa = 8/\log(1 + \sigma_b^{-2})$ with σ_b a uniform bound on the noise. Note that unlike in the Bayesian update equations (4.4) – (4.6), [9] makes distribution-free assumptions on the noise variables ε_t if $q(u; c) \in \mathcal{H}_k(\mathcal{D})$: they can be an arbitrary martingale difference sequence ($\mathbb{E}[\varepsilon_t | \varepsilon_{<t}] = 0$ for all $t \in \mathbb{N}$) uniformly bounded by σ_b.

The exploration-exploitation parameter $\beta_T = 2B + 300\gamma_T \log^3(T/\delta)$ depends on γ_T, the free parameter δ and the bound B. The intuition behind these dependencies is that as the cost function $q(u; c)$ starts to vary more often with respect to the actions u or the contexts c, the RKHS-bound penalizing this function will increase. β_T will increase and (4.7) will end up exploring $\mathcal{U}$ more frequently to track the global minimum of this cost function. We will be more *sure* of finding the global minimum if we decrease δ: this will likewise correspond to an increasing exploration rate as β_T increases.

4.3 Algorithm *TGP*

The scalar cost $q(u; c)$ is a function of inputs and contexts, and learning to minimize this function from scratch can be very impractical in most cases, due to the high-dimensionality of the state space. If we have a nominal model (4.3) at hand however, we can use the costs predicted by the nominal model. Hence the strategy that we follow in *TGP* strikes a different balance between exploration and exploitation as opposed to (4.7): we *exploit* (4.3) from the very start, using the nominal model predicted cost $\hat{q}(u_t; c_t) = d(\hat{x}_{t+1}, s_{t+1})$, a known function of u_t, as a prior. We *explore* only the difference in cost predicted by the nominal model and the actual cost:

$$\delta q_{t+1} = d(x_{t+1}, s_{t+1}) - d(\hat{x}_{t+1}, s_{t+1}) \tag{4.11}$$

With *TGP* we adapt the greedy approach of (4.7) to the reinforcement learning case, and analyze its performance in trajectory tracking. See Figure 4.1 for an illustration. Pseudocode for *TGP* is provided in Algorithm 1. *TGP* at each time stage $t \geq 0$ uses the GP-update equations (4.4) – (4.6) over the joint input-context space $\mathcal{D} = \mathcal{U} \times \mathcal{C}$. It is conditioned on the sets Z_{T-1} and $\mathbf{y}_{T-1}$ acquired through sampling (4.7) at $t = (0, 1, \ldots, T-1)$ with $\hat{q}(u; c)$ added as a known mean function $\mu(z)$ in (4.4).

TGP is an episodic algorithm: the system will continue at each episode ℓ to track the

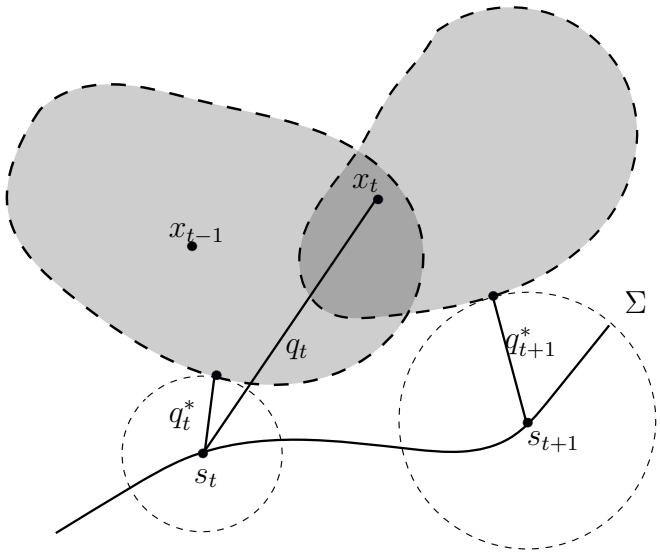


Figure 4.1: Illustrating trajectory tracking: x pursuing s. The gray areas depict the *reachable* regions from the states x_{t-1} and x_t.

Algorithm 1 *TGP*

Input: $\epsilon_s > 0$, $\hat{q}(z)$, $k(z, z')$, $\Sigma = \{s_0, s_1, \ldots, s_N\}$
Initialize $\ell = 1$, $T = 0$, $Z = \emptyset$, $\mathbf{y} = \emptyset$
repeat
  Initialize $t = 0$, $x_{0\ell} = s_0$
  repeat
    $c_{t\ell} \leftarrow (x_{t\ell}, s_{t+1})$
    $u_{t\ell} \leftarrow \arg\min_{u \in U} \mu_T(u; c_{t\ell}) - \beta_T^{1/2} \sigma_T(u; c_{t\ell})$
    $Z \leftarrow Z \cup (u_{t\ell}; c_{t\ell})$
    Observe $\delta q_{(t+1)\ell} = q_{(t+1)\ell} - \hat{q}_{(t+1)\ell}$
    $\mathbf{y} \leftarrow \mathbf{y} \cup \delta q_{(t+1)\ell}$
    $t \leftarrow t + 1$
  until $d(x_{t\ell}, s_t) \geq \epsilon_s$
  $T \leftarrow T + t$
  $\ell \leftarrow \ell + 1$
until $t = N$

given trajectory to a maximum of N time stages and then will reset back to its initial state, ready to embark on the next episode $\ell + 1$. Furthermore *TGP* continues with an episode ℓ as long as the system's trajectory stays within a certain threshold ϵ_s of the desired state, i.e., $d(x_{t\ell}, s_t) < \epsilon_s$. If the trajectory exceeds this bound, i.e. $d(x_{t\ell}, s_t) \geq \epsilon_s$, the system is reset to its initial state. The stopping time is denoted by τ and the total number of time steps (throughout the episodes) by T. To differentiate the episodes, we add a further subscript to x, u, c and q. For example, the state at time stage t in episode ℓ is shown by $x_{t\ell}$.

In order to start *TGP* we need to estimate the GP hyperparameters using any gathered

data. Trial trajectory runs can be used to gather the necessary evaluations for hyperparameter estimation: contexts c, actions u, and costs q. Actions in these trial runs are the feed-forward control inputs calculated using a nominal model. See [13] for feed-forward control signal generation of a differential flat system using splines. With maximum likelihood, we can estimate the hyperparameters θ of a specified kernel structure $k(z, z')$. For example, if the kernel is squared exponential:

$$k(z, z') = \sigma_s^2 \exp^{-1/2(z-z')^{\mathrm{T}}\Theta_z^{-1}(z-z')}$$

then $\theta = (\sigma_s, \vartheta_c, \vartheta_u) \in \mathbb{R}^{2n+m+1}$, where for the diagonal block matrix $\Theta_z = \left(\begin{smallmatrix} \Theta_c & 0 \\ 0 & \Theta_u \end{smallmatrix}\right)$ $\vartheta_c = \mathrm{diag}(\Theta_c)$ and $\vartheta_u = \mathrm{diag}(\Theta_u)$.

Analysis

In this subsection, we prove the convergence of the proposed algorithm *TGP* under the following three assumptions: trackability, controllability, and smooth perturbations of the nominal model.

Trackability. We can view trajectory tracking as tracking a point s that moves with a certain input sequence $v(t)$ on Σ. We assume that the trajectory Σ is *trackable*, i.e. $\forall t\ \exists v(t) \in \mathcal{U}$ s.t. $s_{t+1} = f(s_t, v(t))$. We believe this is a fair assumption to make, otherwise no algorithm can come up with an input sequence $v(t)$ to track Σ. In general a trajectory generation algorithm has access to the nominal model (4.3) and the trajectory generation must be performed *robustly* so that the system can still follow it under the perturbed model (4.1).

Furthermore, we assume that we can start the system at $s_0 \in \Sigma$, i.e. $q_{0\ell} = 0\ \forall \ell$. Otherwise, the trajectory Σ must be recomputed to accommodate for the variation in x_0.

Controllability. The *stopping time* $\tau(\ell)$ for every episode ℓ denotes, as previously introduced, the time stage when the cost function exceeds a certain $\epsilon_s > 0$:

$$\begin{aligned} q_{1\ell} &= r_{1\ell} \\ q_{2\ell} &= r_{1\ell} + r_{2\ell} - \lambda(x_{1\ell}) \\ &\vdots \\ q_{\tau\ell} &= \sum_{i=1}^{\tau(\ell)} \big(r_{i\ell} - \lambda(x_{(i-1)\ell})\big) \geq \epsilon_s \end{aligned} \tag{4.12}$$

where we define the maximum decrease of the cost function as:

$$\lambda(c_t) = q_t - q_{t+1}^* = q_t - \min_{u \in \mathcal{U}} q_{t+1}(u; c_t) \tag{4.13}$$

As noted in section 4.2 we can only prove the convergence of a subsequence of inputs.

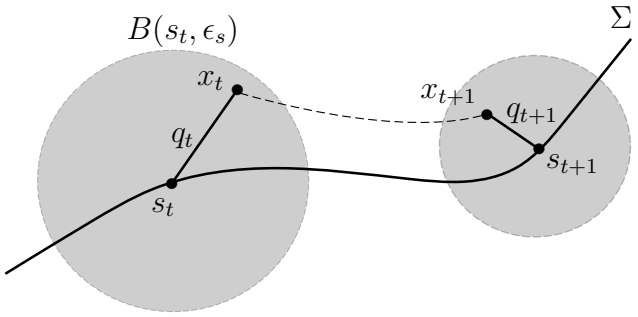


Figure 4.2: Illustrating the controllability assumption: for all states $x_t \in B(s_t, \epsilon_s)$ there exists a u such that x_{t+1} does not move away from s_{t+1} as a result of the nonexpansive mapping, i.e. $q_{t+1} \leq q_t \leq \epsilon_s < \epsilon_{max}$.

We assume this to be nonnegative throughout the state space: $\lambda(c) \geq 0, \forall x \in B(s_t, \epsilon_{max})$ where we take $\epsilon_{max} > 0$. This is an important assumption that lets us focus exclusively on the next desired state. One way to relate it to control theory is to assert that the cost functions $q(u; c)$ are discrete-time *weak* Control Lyapunov Functions for all $x_t \in B(s_t, \epsilon_{max})$, i.e.

$$\min_{u \in \mathcal{U}} q_t - q_{t+1}(u; x_t, s_{t+1}) \leq 0 \tag{4.14}$$

Geometrically this means that the dynamics (4.1) must be a nonexpansive mapping with respect to the semimetric defining the cost function. That is, $\forall x_t \in B(s_t, \epsilon_{max})$, $\exists \phi_t(x_t) \in \mathcal{U}$ s.t. $\phi_t(s_t) = v(t)$:

$$d(x_{t+1}, s_{t+1}) \leq L(u)\, d(x_t, s_t) \tag{4.15}$$

where $L(u) \leq 1$ for $u = \phi_t(x_t)$. Note that both enforce the *relaxed* condition: $\lambda(c) \geq 0$. Inequality is not needed since we assume that we can start at $x_0 = s_0$. See Figure 4.2 for an illustration of the nonexpansive mapping principle. The *contractability* of the balls $B(s_t, \epsilon_s)$ for any $\epsilon_s < \epsilon_{max}$ around the trajectory points s_t imply the existence of a control input u, for every state x_t inside these balls, which will bring the state closer to the next desired state $s_{t+1} \in \Sigma$.

Smooth perturbations. We assume the cost function difference δq is sufficiently smooth: its RKHS-norm must be bounded in order to put a (sublinear) bound on the (cumulative) regret: $\|\delta q(u; c)\|_k^2 \leq B$ for some $B < \infty$ where $k(z, z')$ is any valid combination of linear and squared exponential kernels. This assumption directly carries over from [9]: the sublinear regret bound for *minimizing* δq holds if

$$\Pr\{\forall T, \|\mu_T - \delta q\|_{k_T}^2 \leq \beta_{T+1}\} \geq 1 - \delta \tag{4.16}$$

for $\beta_T = 2\|\delta q\|_k^2 + 300\gamma_T \log^3(T/\delta)$ after sampling Z_T by following (4.7) T times. The bounds in (4.16) depend on γ_T which quantifies the worst-case-scenario for GP-optimization and hence they still hold for *TGP* after sampling Z'_T, different from Z_T due to the addition of the nominal model predicted cost $\hat{q}(u;c)$ as a known function of inputs and contexts to (4.7).

Under these assumptions we have the following result:

THEOREM 4.1 Let $s_t \in \Sigma$ be trackable for $t = 1, \ldots, N$. If $\lambda(c_t) \geq 0$ for some $\epsilon_{max} > 0$ around Σ and $\|\delta q(u;c)\|_k^2 \leq B$ for any valid combination $k(z,z')$ of linear or squared exponential kernels then the following holds with high probability: $\forall \epsilon > 0,\ \exists \ell \in \mathbb{N}\ s.t.\ J_\ell \leq \epsilon$. ■

Proof. Let L be the total number of episodes. The total number of time stages is denoted by $T = \sum_{\ell=1}^{L} \tau(\ell)$. Picking any positive $\epsilon_s < \epsilon_{max}$ we can rewrite (4.12) for a fixed ℓ as:

$$\sum_{i=1}^{\tau(\ell)} r_{i\ell} \geq \epsilon_s + \underbrace{\sum_{i=0}^{\tau(\ell)-1} \lambda(c_{i\ell})}_{=:\Lambda_\ell} \tag{4.17}$$

$$\underbrace{\sum_{\ell=1}^{L(T)} \sum_{i=1}^{\tau(\ell)} r_{i\ell}}_{R_T} \geq \epsilon_s L + \sum_{\ell=1}^{L(T)} \Lambda_\ell \tag{4.18}$$

Using the sublinearity of cumulative regret, and taking the limit:

$$\sum_{\ell=1}^{K(T)} \Lambda_\ell + \epsilon_s L \leq R_T \leq \sqrt{\kappa T \beta_T \gamma_T} \tag{4.19}$$

$$\lim_{T\to\infty} \frac{\sum_{\ell=1}^{L(T)} \Lambda_\ell + \epsilon_s L}{T} = 0 \tag{4.20}$$

holds with high probability $p \geq 1 - \delta$. Assume that as $T \to \infty$, $\tau(\ell)$ never exceeds N. Then $L \geq T/N$. Since $\Lambda_\ell \geq 0,\ \forall \ell = 1, \ldots, L$, we have:

$$\frac{\sum_{\ell=1}^{L(T)} \Lambda_\ell + \epsilon_s L}{T} \geq \frac{\epsilon_s}{N} \tag{4.21}$$

which when taken the limit $T \to \infty$ contradicts (4.20). Hence if (4.10) holds there must be an episode ℓ where $\tau(\ell)$ exceeds N. This means that for every $\epsilon > 0$, taking $\epsilon_s < \min(\epsilon/N, \epsilon_{max})$, if we try long enough w.h.p. there will be an episode ℓ where (4.2) drops

below ϵ:

$$J_\ell = \sum_{t=1}^{N} q_{t\ell} \leq N\frac{\epsilon}{N} = \epsilon \tag{4.22}$$

□

Furthermore, the following proposition gives a bound on the total time stage T when $J_\ell \leq \epsilon$.

Proposition 4.3.1
Under the same assumptions as in Theorem 4.1, the following holds with high probability: $\forall \epsilon > 0,\ \exists \ell \in \mathbb{N}\ s.t.\ J_\ell \leq \epsilon$ before $T \leq T_{max} = \left(\frac{N\alpha}{\epsilon_s}\right)^4$ where $\alpha < \infty$ s.t. $\beta_T \gamma_T \leq \alpha T^{1/4}$. □

Proof. See Appendix A □

Transfer Learning

Transfer learning is the transfer of knowledge from one domain or *problem* to another. Transfer learning is one of the advantages that a GP-based optimization algorithm such as *TGP* has over more conventional learning methods like ILC, where the dynamics are linearized around a trajectory. After linearization, the geometric structure of the differential equation is often lost. If the disturbance dynamics is sufficiently smooth over the state space of different trajectories, or the trajectories are sufficiently close, we expect *TGP* to transfer learned dynamics between similar contexts.

To investigate the degree of transfer in this setting, consider two different cases a and b. In case a a learning algorithm runs on trajectory Σ_1 for a duration of T_1 steps, sampling $Z_{T_1} = \{z_1, \ldots, z_{T_1}\}$ and then continuing on a different trajectory Σ_2 for T_2 steps; while in case b, it runs *from scratch* on Σ_2 for T_2 steps.

As a way to measure the performance gain with *TGP*, we could look at the reduction in cumulative regret for this problem: $R_{T_2} - R_{T_2|Z_{T_1}}$ where $R_{T|Z}$ stands for the cumulative regret after conditioning on observations Z_{T_1} before $t = 0$. However we only have sublinear regret bounds and can not quantify this degree of transfer any further. Instead we propose to look at the bounds of cumulative regret, which leads us to the following proposition:

Proposition 4.3.2
The sublinear bounds $\mathcal{B}_T(Z) = \sqrt{\kappa T \beta_{T|Z} \gamma_{T|Z}}$ bounding the cumulative regret $R_{T|Z}$ after sampling $Z \subset \mathcal{D}$ are nonincreasing, i.e., for any $Z \subset Z'$, $\mathcal{B}_T(Z) \geq \mathcal{B}_T(Z')$. □

Proof. The cumulative regret is bounded as $R_{T|Z} \leq \mathcal{B}_T(Z) = \sqrt{\kappa T \beta_{T|Z} \gamma_{T|Z}}$ where:

$$\begin{aligned}\gamma_{T|Z} &= \max_{A \subset \mathcal{D} \setminus Z : |A| = T} \mathcal{I}(\mathbf{y}_A; q | Z) \\ &= \max_{A \subset \mathcal{D} \setminus Z : |A| = T} H(\mathbf{y}_A | Z) - H(\mathbf{y}_A | q),\end{aligned}$$

$\mathcal{I}(\mathbf{y}_A; q|Z)$ is the *conditional* information gain due to sampling A after sampling Z. Now since $H(\mathbf{y}_A|Z) \geq H(\mathbf{y}_A|Z')$ for Gaussians when $Z' \supset Z$, $\gamma_{T|Z} \geq \gamma_{T|Z'}$. This means that $\mathcal{B}_T(Z)$, monotonically increasing with respect to γ_T, is also nonincreasing. □

In our previous problem formulation, with the help of this proposition we can immediately infer that the transfer gain from case a to case b is nonnegative, i.e. $\mathcal{G}_{ab} = \mathcal{B}_{T_2}(Z_{T_1}) - \mathcal{B}_{T_2}(\emptyset) \geq 0$.

4.4 Experimental Results

In this section we evaluate the performance of the proposed algorithm using a two dimensional model of a quadrotor platform, where the task is to follow predefined trajectories. After defining the numerical model, we first look at the performance of the proposed algorithm when there is a gravity mismatch. In the next step, we investigate knowledge transfer between two different trajectories. Note that for hyperparameter estimation, we use five different wave trajectories and the corresponding feed-forward control signals as examples. These examples are then discarded to avoid overfitting.

Numerical Model

As an example consider the two-dimensional model of a quadrotor given by [13]:

$$\begin{aligned}\ddot{y} &= -f_{\text{coll}} \sin\phi \\ \ddot{z} &= f_{\text{coll}} \cos\phi - g \\ \dot{\phi} &= \omega_x\end{aligned} \tag{4.23}$$

where state $x = (y, \dot{y}, z, \dot{z}, \phi)$. The states y, z are trajectories to be tracked, corresponding to the horizontal and vertical axes and the control input $u = (f_{\text{coll}}, \omega_x)$, where $f_{\text{coll}} = \sum_{i=1}^{4} f_i$ is the collective thrust and ω_x is rate of change of the angle of attack w.r.t. the x-axis. Input constraints are given by:

$$\begin{aligned}f_{min} \leq f_i \leq f_{max}, \\ |\dot{f}_i| \leq \dot{f}_{max},\end{aligned}$$

Source code for generating the experimental results can be downloaded from bitbucket/icml_tgp.

$$|\dot{\phi}| \leq \dot{\phi}_{max},$$
$$|\ddot{\phi}| \leq \ddot{\phi}_{max}.$$

The values used throughout the numerical examples are given in the Appendix in Table A.1. Unmodeled dynamics in quadrotors could be due to parameter mismatch (e.g. gravity difference) or a more general repeating disturbance (e.g. a fan). See Appendix for two examples.

Learning under Model Mismatch

Here we show the *TGP* learning results for the quadrocopter operating under a gravity mismatch. The gravity is taken to be $g_{\text{uranus}} = 10.5$, but the nominal dynamics is assuming earth gravity, $g_{\text{earth}} = 9.81$. We model the cost function (A.7) as having bounded RKHS-norm, $\|q(u;c)\|_k^2 < B$ under the following covariance function:

$$\begin{aligned} k(z,z') &= k_u(u,u')k_c(c,c') + \sigma_n^2\delta_{aa'} \\ k_u(u,u') &= u^{\mathrm{T}}\Lambda_u^{-2}u' \\ k_c(c,c') &= \sigma_s^2\exp(-\frac{1}{2}(c-c')^{\mathrm{T}}\Lambda_c^{-2}(c-c')) \end{aligned}$$

where $z = (u;c)$. Diagonal matrices Λ_u and Λ_{c_i} transform anisotropic coordinates into isotropic coordinates or they can be motivated from Automatic Relevance Determination (ARD) point of view where Λ_u^2 and Λ_c^2 encode the relevance of inputs and contexts, respectively [11]. The bound on the noise, σ_b was set to 0.15 during the simulations.

In Figure 4.3 we compare the performance of *TGP* with ILC and (nonlinear) MPC with horizon $M = 5$. Weighted Sum of Squares (SSE) errors in (4.2) are plotted to show learning during the first 6 episodes. A weighted squared Euclidean distance is used as the cost function where the diagonal matrix with entries $(1, 1, 1, 1, 0.01)$ is taken as the weighting matrix Q. The best results for the three algorithms during the episodes are shown in Table 4.1. Figure 4.3 clearly shows that the method can outperform more conventional methods like MPC when disturbances in the form of unknown dynamics are present, and can compare favorably with feedforward learning controllers like ILC. The yz-trajectories followed during these episodes are plotted in Figure 4.4.

Table 4.1: SSE errors

Method / Episode No.	SSE
MPC, horizon = 5	0.0617
ILC, episode 6	0.0306
TGP, episode 6	0.0271

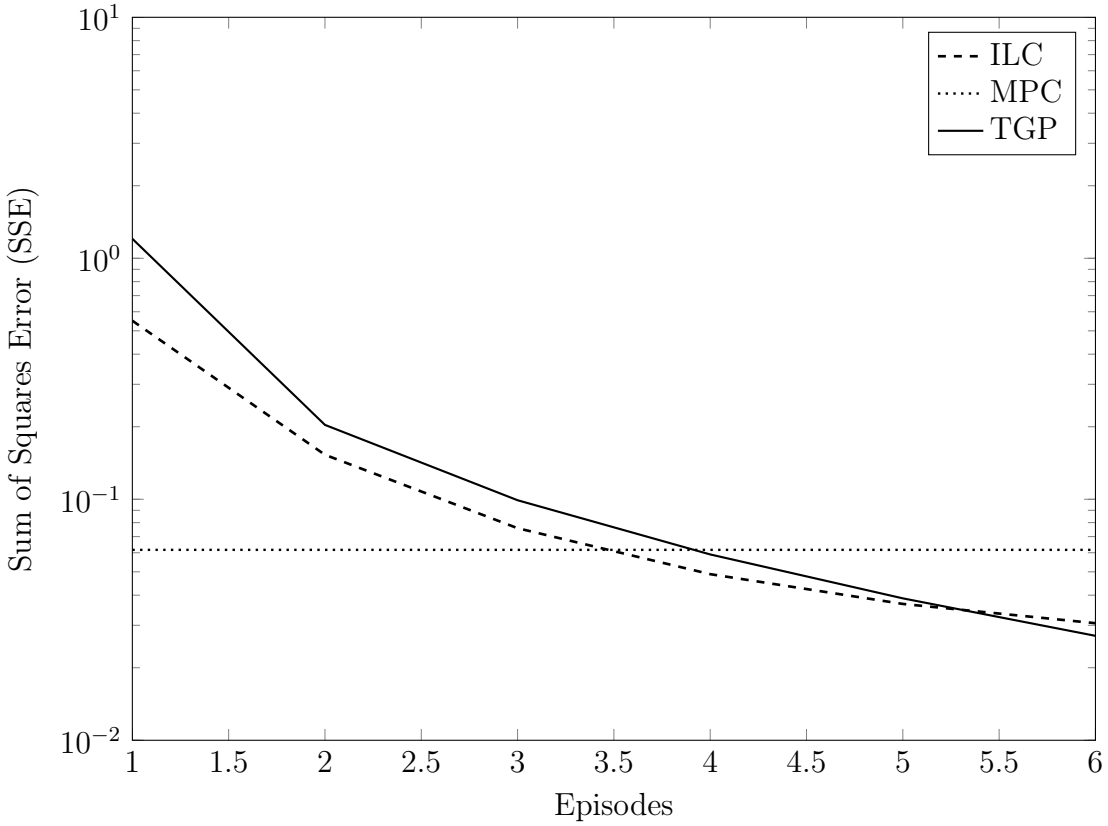


Figure 4.3: Comparison of *TGP*, ILC and MPC.

Transfer Learning

Here we show the transfer learning performance of *TGP*. We implement the scenario considered in section 4.3 three times using three random wave trajectories: we first run *TGP* on an initial fixed wave trajectory Σ_0 and then switch to a different trajectory Σ_i, $i \in 1, 2, 3$. Figure 4.5 shows the tracking error of Σ_i under the transfer learning setting. We compare with TGP running with no prior knowledge and with ILC.

4.5 Conclusion

In this paper, we have proposed and analyzed *TGP*, a GP-optimization based RL-algorithm, demonstrating its effectiveness in learning unknown dynamics and knowledge transfer between different contexts. We have validated its performance in numerical examples. Unknown dynamics, if severe, can prevent more conventional methods such as Model Predictive Control or Iterative Learning Control from tracking any given trajectory. *TGP* on the other hand learns to track online a scalar cost function, and converges to a given trajectory under mild assumptions. It works by solving the exploration-exploitation problem: it *explores* new control inputs when the uncertainty of that input is high enough,

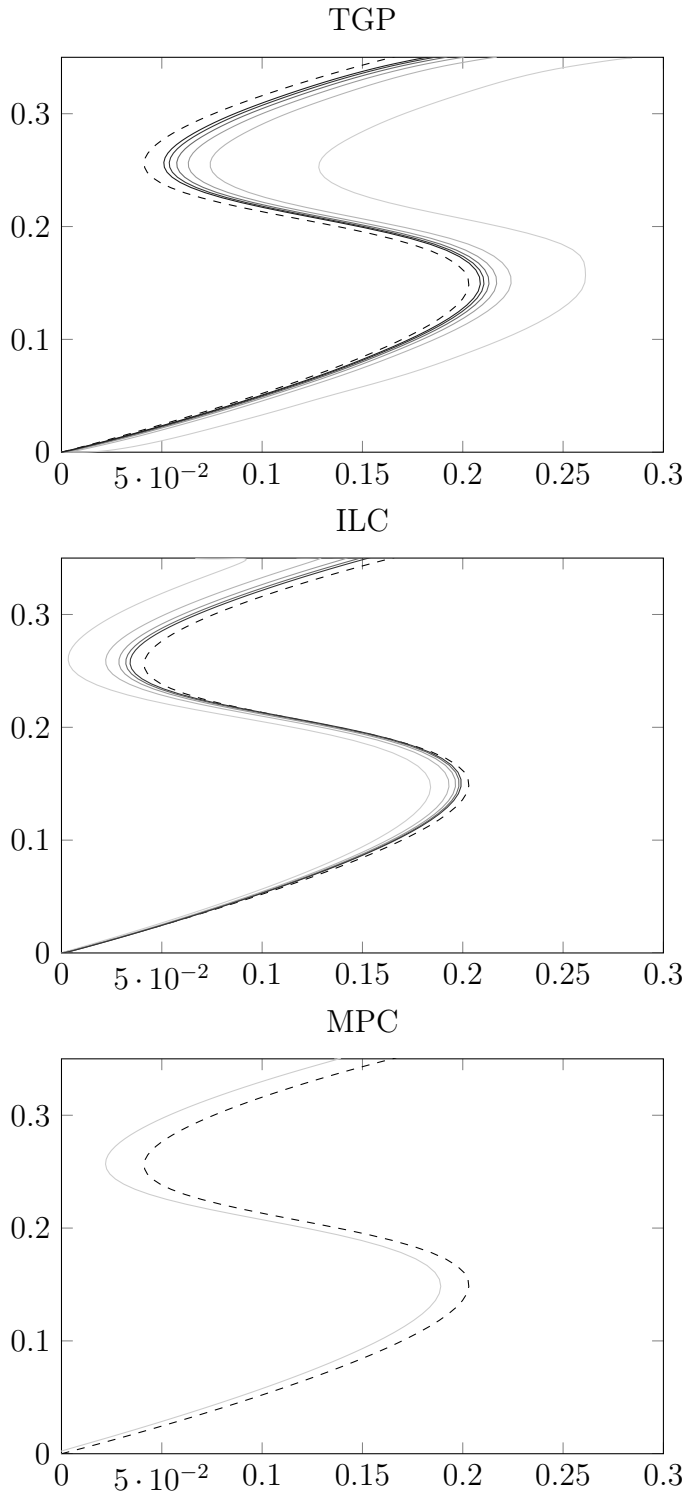


Figure 4.4: Trajectories $\{(y_t, z_t)\}$ followed by a quadrotor under *TGP*, ILC and MPC: the dashed lines represent the trajectory that needs to be tracked. Solid lines represent the trajectories of the quadrotor tracked by the algorithms. Increasing shades of gray represent increasing episodes.

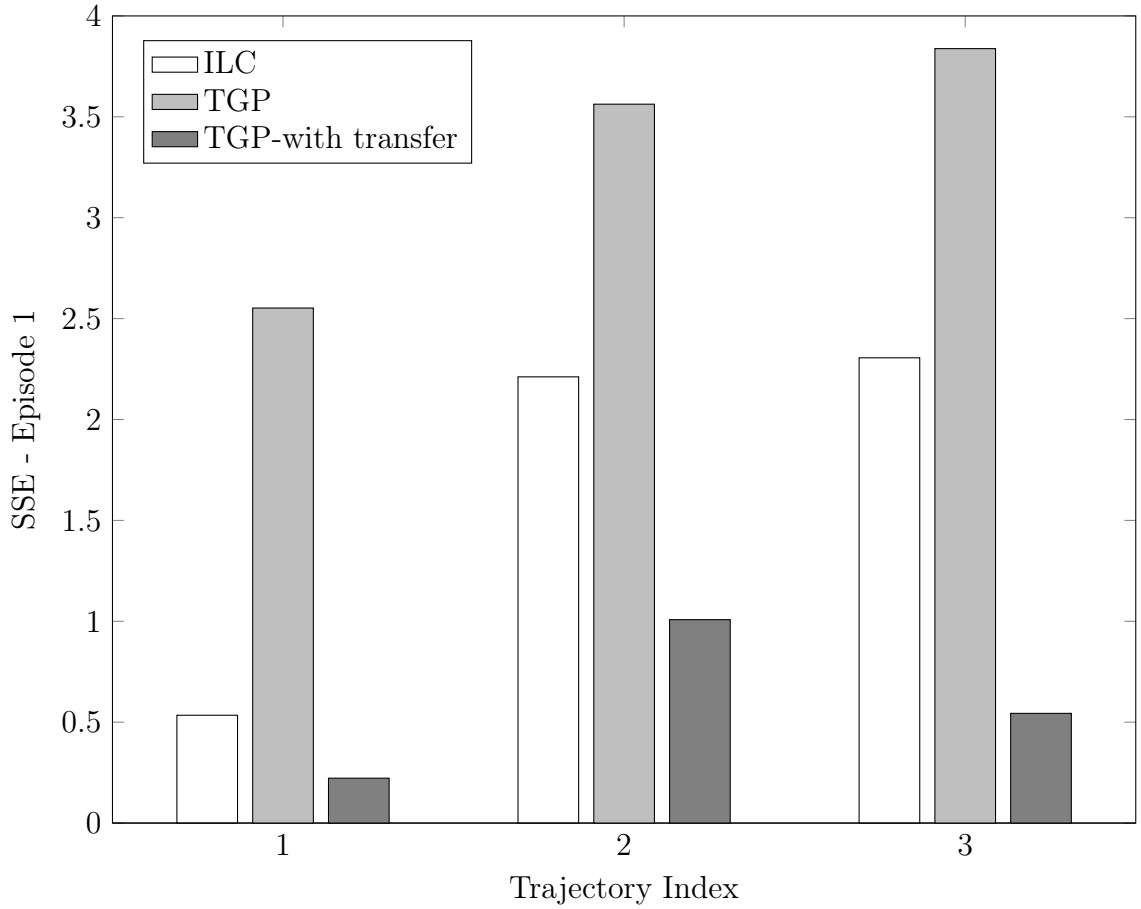


Figure 4.5: Comparison of ILC, *TGP* with and without transfer (from scratch). In the case of TGP *with transfer* the algorithm was first run on a different trajectory.

and it *exploits* the learned dynamics as it gets better at predicting the cost function.

The sublinear regret proofs presented in [8, 9] hold only when the hyperparameters of the GP from which the function to be optimized is drawn are known. In practice estimation of hyperparameters can be performed using Maximum Likelihood and its variants but we are currently unaware of any results on the sensitivity analysis. We have evidence to believe that under mild mismatch only the speed of convergence is affected, however changes in the kernel structure can hinder learning. This leads to the problem of adaptive estimation of the covariance functions [14], which will play an increasing role in learning under unpredictable environments, such as those studied in RoboEarth [2].

References

[1] D. Bristow, M. Tharayil, and A. Alleyne, "A survey of iterative learning control," *Control Systems, IEEE*, vol. 26, no. 3, pp. 96 – 114, june 2006.

[2] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft, "RoboEarth," *Robotics & Automation Magazine, IEEE*, vol. 18, no. 2, pp. 69–82, 2011.

[3] J. Ko, D. Klein, D. Fox, and D. Haehnel, "Gaussian processes and reinforcement learning for identification and control of an autonomous blimp," in *Robotics and Automation, 2007 IEEE International Conference on*, april 2007, pp. 742 –747.

[4] M. P. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *In Proceedings of the International Conference on Machine Learning*, 2011.

[5] M. P. Deisenroth, C. E. Rasmussen, and D. Fox, "Learning to control a low-cost manipulator using data-efficient reinforcement learning," in *Robotics: Science and Systems*, 2011.

[6] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," *Journal of Global Optimization*, vol. 21, pp. 345–383, 2001.

[7] D. Lizotte, T. Wang, M. Bowling, and D. Schuurmans, "Automatic gait optimization with gaussian process regression," in *in Proc. of IJCAI*, 2007, pp. 944–949.

[8] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process bandits without regret: An experimental design approach," *CoRR*, vol. abs/0912.3995, 2009.

[9] A. Krause and C. S. Ong, "Contextual gaussian process bandit optimization," in *NIPS'11*, 2011, pp. 2447–2455.

[10] G. Wahba, *Spline Models for Observational Data*. SIAM, 1990.

[11] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[12] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 1st ed. Springer, 2007.

[13] A. P. Schoellig, F. L. Mueller, and R. D'Andrea, "Optimization-based iterative learning for precise quadrocopter trajectory tracking," *Autonomous Robots*, vol. 33, pp. 103–127, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10514-012-9283-2

[14] D. Ginsbourger, C. Helbert, and L. Carraro, "Discrete mixtures of kernels for kriging-based optimization," *Quality and Reliability Engineering International*, vol. 24, no. 6, pp. 681–691, 2008. [Online]. Available: http://dx.doi.org/10.1002/qre.945

5

The Cubli

5.1 Introduction

Inverted pendulum systems have a very rich history [1] and have been widely used to test, demonstrate and benchmark new control concepts and theories [2]. Furthermore, development of new control algorithms for the pendulum system itself is still an active area of research [3]–[5].

The Cubli is a 3D inverted pendulum when balancing on its corner, see Fig. 5.1, and it becomes a 1D pendulum when balancing on its edge. The Cubli has two features that set it apart from other 3D inverted pendulum testbeds [6, 7]: one is its relatively small footprint - hence the name "Cubli," which is derived from the Swiss German diminutive for "cube"; the other is its ability to jump up from a resting position without any external support - a feature that not only poses an interesting engineering challenge, but also serves as a compelling demonstration of mechanics and control concepts for the general public.

Fig. 5.2 shows the Cubli's jump-up strategy. The Cubli begins lying flat on one of its faces. It then accelerates and rapidly brakes one of its reaction wheels, allowing it to jump up to one of its edges. Once it is balancing on an edge, the Cubli accelerates and then rapidly brakes its next two reaction wheels, allowing it to jump up on its corner. Now in the corner-balancing position, the Cubli applies controlled torques to its reaction wheels in order to maintain this upright state.

Reaction wheel-based internal actuation for micro gravity environments was first presented in [8] and further investigated in [9] for planetary exploration in the absence of strong traction forces [10]. Although motor torques are sufficient for overcoming the weight of a device in microgravity environments like outer space, large amounts of torque are necessary to do so within earth's gravitational pull. The Cubli achieves this by spinning its reaction wheels at high angular velocities and then rapidly braking them. This rapid braking produces torques that are well beyond the limitation of the motors, and

This paper is submitted to *Mechatronics (Elsevier)*, March 2014.

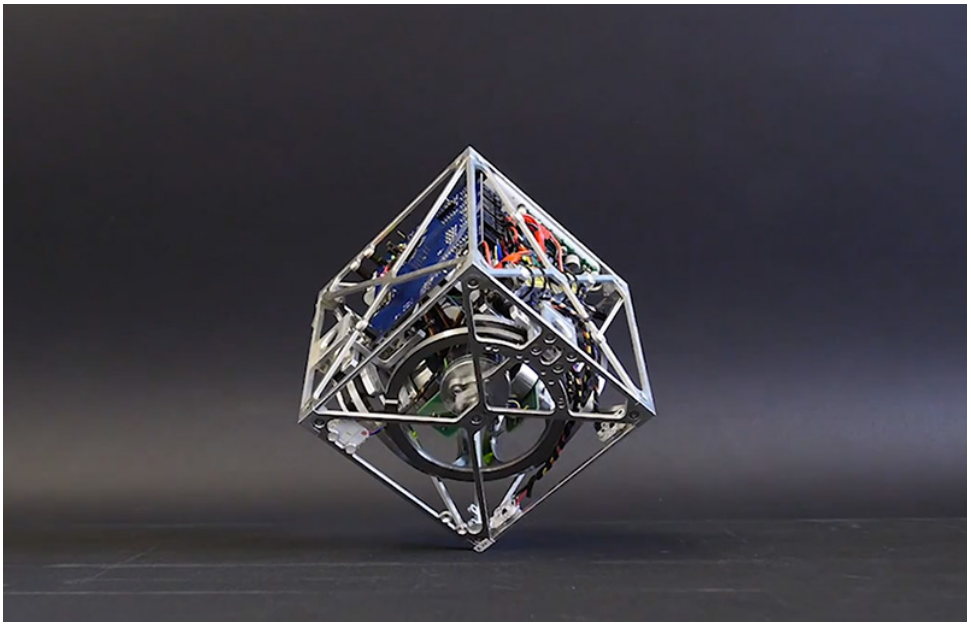


Figure 5.1: Cubli balancing on its corner. Every 20 ms, measurements from the 6 inertial measurement units (IMUs) are fused to get an estimate of the state and based on this estimate the necessary reaction torques to balance are calculated and sent to the motor controllers.

are what allow the Cubli to actuate internally - or self-actuate - without any external support. This concept of braking was recently used for self-assembly in the M-blocks [11] modular robot platform.

5.2 Mechatronic Design

The three-dimensional Cubli design started off with the following question:

> *How to build a* 15 cm *sided cube that can jump up and balance on its corner, using off-the-shelf motors, batteries, and electronic components?*

Mechanical Design

The Cubli's mechanical design was dominated by the need for structural rigidity and off-the-shelf components. Fig. 5.3 shows a CAD model of the Cubli, which consists of aluminum housing (denoted by h in the following sections) that holds the three reaction wheels and their respective motors. Although light-weight housing would result in high recovery angles during balancing, the housing must be strong enough to withstand large forces/torques during rapid braking.

The required angular velocity for jump-up can be reduced by increasing the wheel inertia. Since the wheel size is constrained, increasing the wheel inertia would result

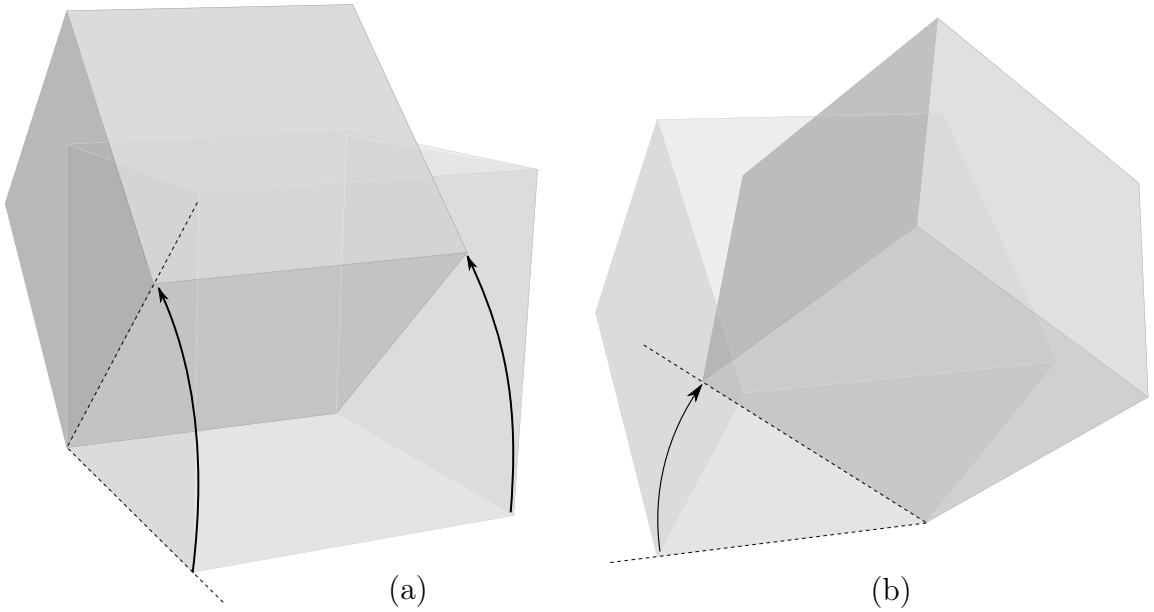


Figure 5.2: The Cubli's jump-up strategy. (a) Flat-to-Edge: Initially lying flat on its face, the Cubli jumps up to stand on its edge. (b) Edge-to-Corner: The Cubli goes from balancing on an edge to balancing on a corner.

primarily in a higher wheel mass - ultimately reducing the recovery angles during the balancing phase. Hence the wheel inertia was a trade-off between the recovery angle needed for balancing, and the angular velocity needed for the jump-up. A gear chain between the wheel and the motor was avoided since it would not allow the high angular velocities for jump-up and would add extra weight and volume. Although the balancing maneuver, which requires high torques, was affected by the choice of not including any gears, the brushless DC motors were still able provide enough torque for recovery angles up to 6°. Figure 5.4 shows the latest iteration of the braking mechanism, which is based on bicycle brake pads. An RC servo (stall torque 0.18 Nm) with an elliptic head was used to drive the bicycle brake pad and produced a maximum torque of around 3.5 Nm on the reaction wheels.

Electrical Design

A simplified diagram of the electronics is presented in Fig. 5.5. The STM32 discovery board (ARM7 Cortex-M4, 168 MHz) from STMicroelectronics is used as the Cubli's main controller. Six IMUs (MPU-6050, InvenSense), attached to the Cubli housing, are used to estimate the tilt and angular velocity of the housing. Each IMU consists of a rate gyro and an accelerometer, and is connected to the main controller through the I2C bus. A 50 W brushless DC motor, EC-45-flat, from Maxon Motor AG is used to drive the reaction wheels. The three brushless DC motors are controlled by three DEC 36/2 modules, digital four quadrant brushless DC motor controllers. The motor controller and main controller communicate over the CANopen protocol. PWM signals drive the breaking mechanism's

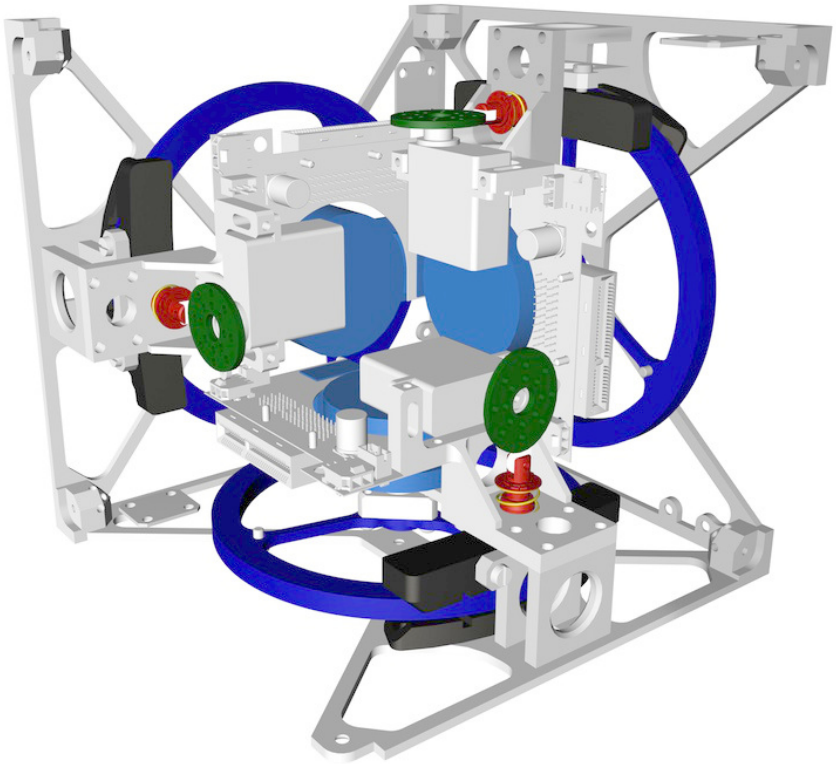


Figure 5.3: CAD drawing of the Cubli, with three of its faces and some of its electronics and wiring removed in order to illustrate.

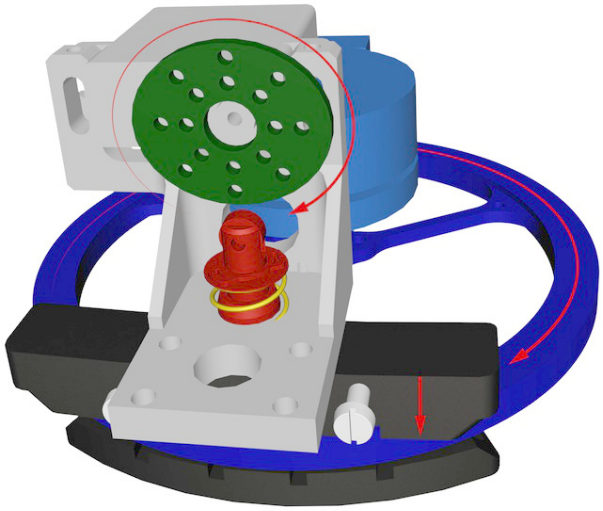


Figure 5.4: The latest iteration of the braking mechanism, which is based on bicycle brakes. An RC servo with an elliptic head drives the top brake pad. Due to the slight play in the reaction wheel's bearing, the wheel tilts and touches the (static) bottom pad when braking, thus increasing the braking friction.

RC Servo (HSG-5084MG), see Fig. 5.4.

The FreeRTOS scheduler's STM32 port was used in the software framework to multitask the estimation and control algorithms. A completely open source and a free development environment for embedded systems named (named ODeV [13], and based on the Eclipse IDE) was used for the software development.

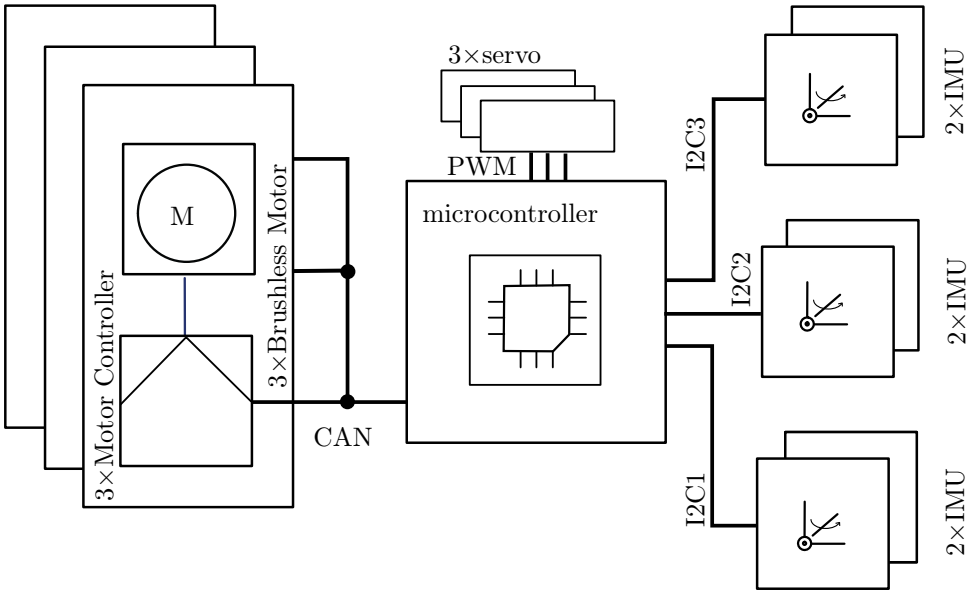


Figure 5.5: The Cubli electronics schematic. Six IMUs (rate-gyro and accelerometer) are connected to the ARM7 microcontroller with three I2C buses. The three motor controllers communicate with the microcontroller over the CAN bus.

5.3 Modelling

Edge Balancing

This subsection presents the modelling of a 1D reaction wheel-based inverted pendulum, i.e., the Cubli balancing on an edge as shown in Fig. 5.6. Due to the similarity of some key properties in both the 1D and 3D cases, this subsection sets the stage for the analysis of the 3D inverted pendulum in the later sections.

Let φ denote the tilt angle of the Cubli's housing and ψ denote angular position of the reaction wheel, see Fig. 5.6. Next, let Θ_w denote the reaction wheel's moment of inertia, Θ_0 denote the system's total moment of inertia around the pivot point in the body fixed coordinate frame, and m_{tot} and l represent the total mass and distance between the pivot point to the center of gravity of the whole system.

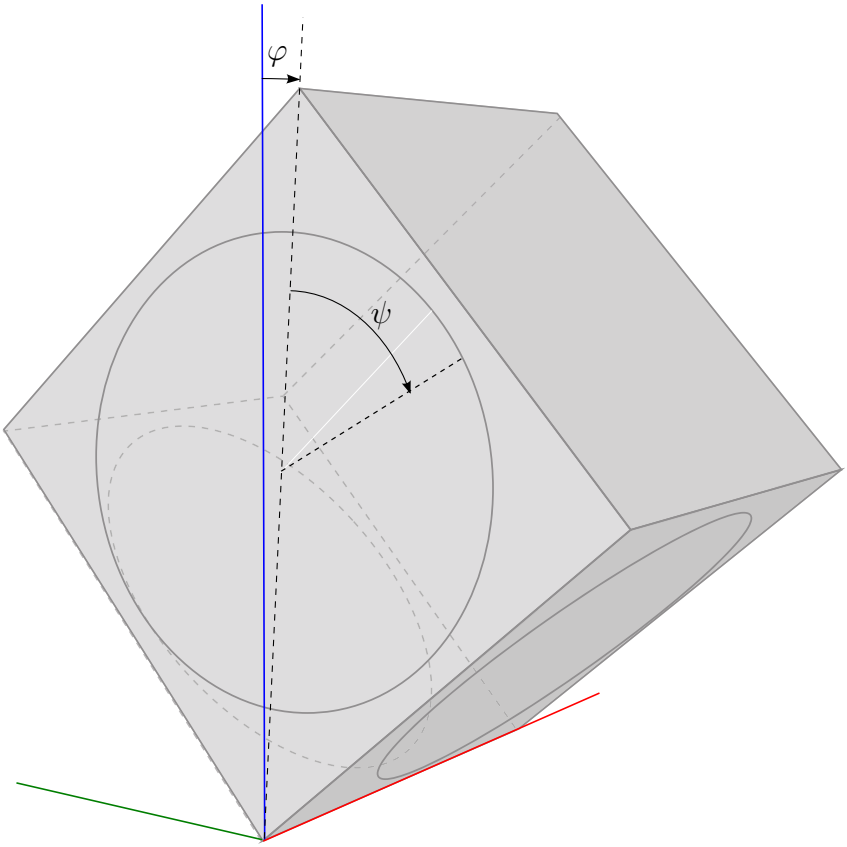


Figure 5.6: Cubli balancing on an edge.

The Lagrangian [12] of the system is given by

$$\mathcal{L} = \frac{1}{2}\hat{\Theta}_0\dot{\varphi}^2 + \frac{1}{2}\Theta_w(\dot{\varphi} + \dot{\psi})^2 - mg\cos\varphi, \tag{5.1}$$

where $\hat{\Theta}_0 = \Theta_0 - \Theta_w > 0$, $m = m_{tot}l$ and g is the constant gravitational acceleration. The generalized momenta are defined as

$$p_\varphi := \frac{\partial\mathcal{L}}{\partial\dot{\varphi}} = \Theta_0\dot{\varphi} + \Theta_w\dot{\psi}, \tag{5.2}$$

$$p_\psi := \frac{\partial\mathcal{L}}{\partial\dot{\psi}} = \Theta_w(\dot{\varphi} + \dot{\psi}). \tag{5.3}$$

Let T denote the torque applied to the reaction wheel by the motor. The equations of motion can be derived using the Euler-Lagrange equations with the torque T as a non-potential force. This yields

$$\dot{p}_\varphi = \frac{\partial\mathcal{L}}{\partial\varphi} = mg\sin\varphi, \tag{5.4}$$

$$\dot{p}_\psi = \frac{\partial \mathcal{L}}{\partial \psi} + T = T. \tag{5.5}$$

Note that the introduction of the generalized momenta in (5.2) and (5.3) leads to a simplified representation of the system, where (5.4) resembles an inverted pendulum augmented by an integrator in (5.5).

Since the actual position of the reaction wheel is not of interest, we introduce $x := (\varphi, p_\varphi, p_\psi)$ to represent the reduced set of states, and describe the dynamics of the mechanical system as follows:

$$\dot{x} = \begin{pmatrix} \dot{\varphi} \\ \dot{p}_\varphi \\ \dot{p}_\psi \end{pmatrix} = f(x,T) = \begin{pmatrix} \hat{\Theta}_0^{-1}(p_\varphi - p_\psi) \\ mg\sin\varphi \\ T \end{pmatrix}. \tag{5.6}$$

Since the measured quantities are $y := (\varphi, \dot{\varphi}, \dot{\psi})$, the output equations are given by

$$y = \begin{pmatrix} \varphi \\ \dot{\varphi} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \hat{\Theta}_0^{-1} & -\hat{\Theta}_0^{-1} \\ 0 & -\hat{\Theta}_0^{-1} & \Theta_w^{-1} + \hat{\Theta}_0^{-1} \end{pmatrix} x =: Cx. \tag{5.7}$$

Linearizing equations (5.6) and (5.7) around the equilibrium $x_{eq} = (0,0,0)$ yields

$$\dot{x} = Ax + Bu \tag{5.8}$$

$$y = Cx, \tag{5.9}$$

with

$$A = \begin{pmatrix} 0 & \hat{\Theta}_0^{-1} & -\hat{\Theta}_0^{-1} \\ mg & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},\ B = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \quad \text{and} \quad u = T.$$

Corner Balancing

This subsection presents the modelling of the Cubli balancing on a corner as shown in Fig. 5.7. Let Θ_0 denote the total moment of inertia of the full Cubli around the pivot point O (see Fig. 5.7), $\Theta_{wi},\ i = 1,2,3$ denote the moment of inertia of each reaction wheel (in the direction of the corresponding rotation axis), and define $\Theta_w := \mathrm{diag}(\Theta_{w1}, \Theta_{w2}, \Theta_{w3})$, $\hat{\Theta}_0 := \Theta_0 - \Theta_w$. Next, let $\vec{m}$ denote the position vector from the pivot point to the center of gravity multiplied by the total mass and let $\vec{g}$ denote the gravity vector. The projection of a tensor onto a particular coordinate frame is denoted by a preceding superscript, i.e. ${}^B\Theta_0 \in \mathbb{R}^{3\times3},\ {}^B(\vec{m}) = {}^Bm \in \mathbb{R}^3$. The arrow notation is used to emphasize that a vector

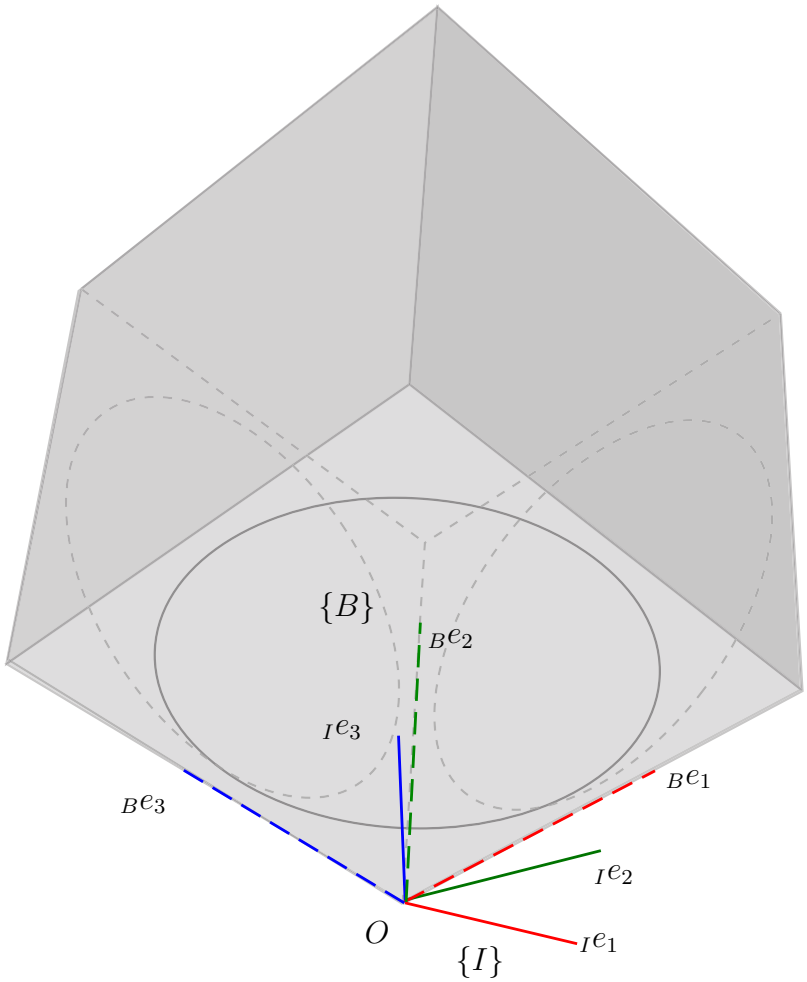


Figure 5.7: Cubli balancing on the corner. ${}_B e_*$ and ${}_I e_*$ denote the principle axis of the body fixed frame $\{B\}$ and inertial frame $\{I\}$. The pivot point O is the common origin of coordinate frames $\{I\}$ and $\{B\}$.

(and tensor) should be *a priori* thought of as a linear object in a normed vector space detached from its coordinate representation with respect to a particular coordinate frame. Since the body fixed coordinate frame $\{B\}$ is the most commonly projected coordinate frame, we usually remove its preceding superscript for the sake of simplicity. Note further that ${}^B\hat{\Theta}_0 = \hat{\Theta}_0 \in \mathbb{R}^{3\times 3}$ is positive definite.

The Lagrangian of the system is given by

$$\mathcal{L}(\omega_h, g, \omega_w, \phi) = \frac{1}{2}{\omega_h}^{\mathrm{T}}\hat{\Theta}_0\omega_h + \frac{1}{2}\,(\omega_h + \omega_w)^{\mathrm{T}} \Theta_w(\omega_h + \omega_w) + m^{\mathrm{T}}g, \tag{5.10}$$

where ${}^B(\vec{\omega}_h) = \omega_h \in \mathbb{R}^3$ denotes the body angular velocity and ${}^B(\vec{\omega}_w) = \omega_w \in \mathbb{R}^3$ denotes the reaction wheel's angular velocity. The components of $T \in \mathbb{R}^3$ contain the torques as applied to the reaction wheels. In the body fixed coordinate frame, the vector m is

constant, whereas the time derivative of g is given by ${}^{B}(\dot{\vec{g}}) = 0 = \dot{g} + \omega_h \times g = \dot{g} + \tilde{\omega}_h g$. The tilde operator applied to the vector $v \in \mathbb{R}^3$, i.e $\tilde{v}$, denotes the skew symmetric matrix for which $\tilde{v}a = v \times a$ holds for all $a \in \mathbb{R}^3$.

Using the generalized momenta defined as

$$p_{\omega_h} =: \frac{\partial \mathcal{L}}{\partial \omega_h}^{\mathrm{T}} = \Theta_0 \omega_h + \Theta_w \omega_w, \tag{5.11}$$

$$p_{\omega_w} =: \frac{\partial \mathcal{L}}{\partial \omega_w}^{\mathrm{T}} = \Theta_w (\omega_h + \omega_w) \tag{5.12}$$

results in the equations of motion given by

$$\dot{g} = -\tilde{\omega}_h g, \tag{5.13}$$

$$\dot{p}_{\omega_h} = -\tilde{\omega}_h p_{\omega_h} + \tilde{m} g, \tag{5.14}$$

$$\dot{p}_{\omega_w} = T. \tag{5.15}$$

Note that there are several ways of deriving the equations of motion; for example in [13] the concept of virtual power has been used. A derivation using the Lagrangian formalism is shown in the appendix of [14] and highlights the similarities between the 1D and 3D case. The Lagrangian formalism also motivates the introduction of the generalized momenta.

Using $\dot{\vec{v}} = \dot{v} + \omega_h \times v$, the time derivative of a vector in a rotating coordinate frame, (5.13)-(5.15) can be further simplified to

$$\dot{\vec{g}} = 0, \tag{5.16}$$

$$\dot{\vec{p}}_{\omega_h} = \vec{m} \times \vec{g}, \tag{5.17}$$

$$\dot{p}_{\omega_w} = T. \tag{5.18}$$

This highlights, in particular, the similarity between the 1D and 3D inverted pendula. Since the norm of a vector is independent of its representation in a coordinate frame, the 2-norm of the impulse change is given by $||\dot{\vec{p}}_{\omega_h}||_2 = ||\vec{m}||_2 ||\vec{g}||_2 \sin\phi$. In this case, ϕ denotes the angle between the vectors $\vec{m}$ and $\vec{g}$. Additionally, as in the 1D case, p_{ω_w} is the integral of the applied torque T.

5.4 State Estimation

The angular velocity of the Cubli's housing ω_{h} is estimated by averaging the six rate-gyro measurements. The estimates of wheel velocity ω_{w} comes directly from the motor

`http://www.idsc.ethz.ch/Research_DAndrea/Cubli/cubliCDC13-appendix.pdf`

controller. This section gives an brief overview of the Cubli's accelerometer-based tilt estimation algorithm developed in [15] for the Balancing Cube [7] project.

A one dimensional tilt estimation using two accelerometers is shown in Fig. 5.8. The Cubli uses six accelerometers to estimate its tilt, or more precisely, to estimate the gravity vector ${}^{B}g$ expressed in its body fixed frame. Each accelerometer is rigidly attached to the Cubli's housing and its positions in the body fixed frame $\{B\}$ is known and denoted by $p_i \in \mathbb{R}^3$, $i = 1, \ldots, 6$.

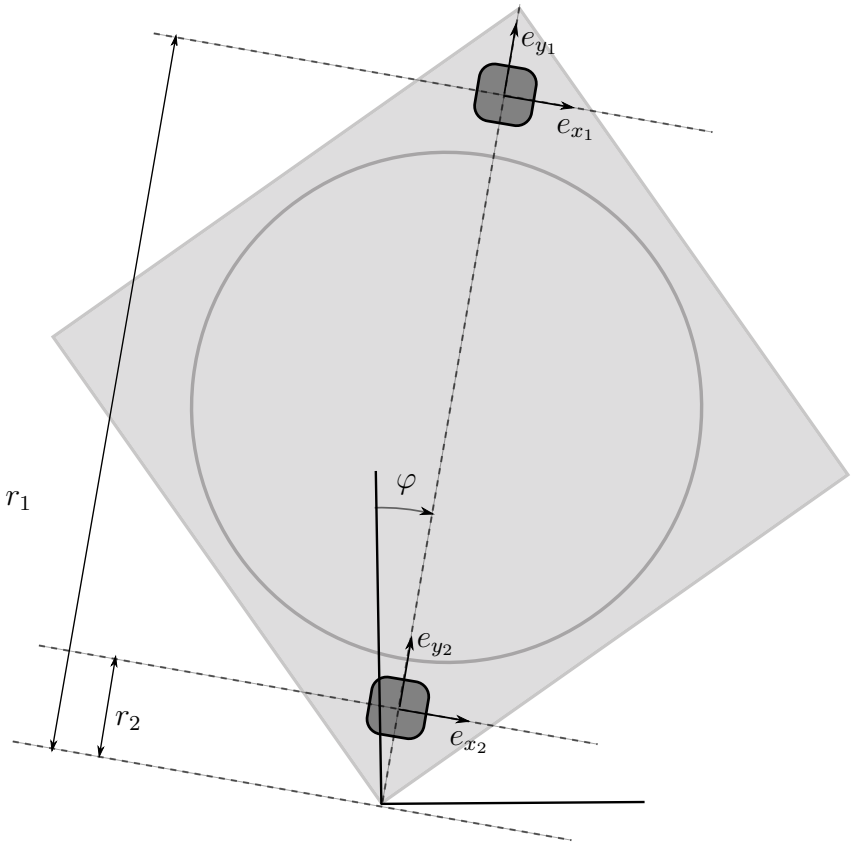


Figure 5.8: 1D tilt (φ) estimation using two accelerometers. The noise free measurement m_i, $i \in 1, 2$ of the accelerometer is given by $(r_i\ddot{\varphi} + g\sin\varphi, -r_i\dot{\varphi}^2 - g\cos\varphi, 0)$. The dynamic terms ($\dot{\varphi}$, $\ddot{\varphi}$) can be eliminated by $m_1 - \mu m_2 = ((1-\mu)g\sin\varphi, -(1-\mu)g\cos\varphi, 0)$ $=: (m_x, m_y)$, where $\mu = r_1/r_2$ and the estimate of the tilt angle is given by $\hat{\varphi} := \tan^{-1}(-m_x/m_y)$.

A noise-free accelerometer measurement ${}^{\hat{A}_i}m_i$ of accelerometer i is given by

$$ {}^{\hat{A}_i}m_1 = {}^{\hat{A}_i}_{B}R\, {}^{B}_{I}R\, ({}^{I}_{B}\ddot{R}\, {}^{B}p_i + {}^{I}g), \tag{5.19} $$

where $\{\hat{A}_i\}$ denotes the local frame of the i^{th} sensor, ${}^{I}g$ is the gravity vector expressed in the inertial frame $\{I\}$, and ${}^{I}_{B}\ddot{R}$ is the second derivative of ${}^{I}_{B}R$ satisfying

$$ {}^{I}\ddot{p}_i = {}^{I}_{B}\ddot{R}\, {}^{B}p_i. \tag{5.20} $$

Now, changing the reference frames to the body fixed frame $\{B\}$ in (5.19) by multiplying both sides by ${}^{B}_{\hat{A}_i}R$ gives

$$ {}^{B}m_i = \tilde{R}\,{}^{B}p_i + {}^{B}g, \tag{5.21} $$

where $\tilde{R} = {}^{B}_{I}R\,{}^{I}_{B}\ddot{R}$.

Using all accelerometer measurements (5.21) can be expressed as a matrix equation of the following form:

$$ M = QP, \tag{5.22} $$

where

$$ \begin{aligned} M &:= \begin{bmatrix} {}^{B}m_1 & {}^{B}m_2 & \cdots & {}^{B}m_6 \end{bmatrix} \in \mathbb{R}^{3\times 6}, \\ Q &:= \begin{bmatrix} {}^{B}g & \tilde{R} \end{bmatrix} \in \mathbb{R}^{3\times 4}, \\ P &:= \begin{bmatrix} 1 & 1 & \cdots & 1 \\ p_1 & p_2 & \cdots & p_6 \end{bmatrix} \in \mathbb{R}^{4\times 6}. \end{aligned} $$

The optimal estimate of Q under noisy measurements, while Q is restricted to linear combinations of measurements M, is given by [15]

$$ \hat{Q} = M\hat{X}, \quad \hat{X} := P^{\mathrm{T}}(PP^{\mathrm{T}})^{-1}. \tag{5.23} $$

Finally, this gives the gravity vector estimate

$$ {}^{B}\hat{g} = M\hat{X}(:,1) \tag{5.24} $$

as a linear combination of measurements M. Note that, since $\hat{X}$ in (5.24) depends only on the sensor positions, it can be calculated offline to improve the speed of the estimation process. In a further step, ${}^{B}\hat{g}$ from (5.24) are fused with the integrated rate-gyro measurements to reduce the noise levels. See [15] for more details.

In contrast to many standard methods for state estimation, the above tilt estimator does not require a model of the system's dynamics; the only required information is the location of the sensors on the rigid body. Since no near-equilibrium assumption is made, the estimator above can be used during both the balancing and the jump-up maneuvers.

5.5 System Identification

This section describes an offline frequency domain-based approach for identifying the parameters of the Cubli. This section focuses on the edge-balancing-based identification procedure, a procedure that excites the Cubli when it is balancing on a edge under a linear

controller. If this identification procedure is repeated for all three axes of the Cubli's body fixed frame $\{B\}$, see Fig. 5.7, all parameters except the off-diagonal entries of $\hat{\Theta}_0$ can be identified.

Edge Balancing

This subsection describes the identification of the parameters

$$\eta_{eb} = (\hat{\Theta}_0, \Theta_w, m)^{\mathrm{T}} \in \mathbb{R}^3 \tag{5.25}$$

of equations (5.8) and (5.9). Let $G(s, \eta_{eb})$ denote the parametric transfer function from the input u to the output y, given by

$$G(s, \eta_{eb}) = C(sI - A)^{-1}B. \tag{5.26}$$

While balancing on its edge, the Cubli is excited by the following random phase multisine signal:

$$r(t) = \sum_{k=1}^{F} A_k \sin(2\pi f_0 k t + \chi_k), \quad A_k \in \mathbb{R}^+, \tag{5.27}$$

where the signal is sampled with 50 Hz and χ_k is uniformly distributed in $[0, 2\pi)$. A realization is defined by the set $\{\chi_k\}$, $k \in \{1, 2, ..., F\}$, i.e. F draws from a uniform distribution. The identification experiment of the Cubli is set up by choosing $F = 20$, $f_0 = 0.2$ Hz and $A_k = 10$ for $k \in \{1, 2, ..., F\}$, such that frequencies from 0.2 up to 4 Hz are excited. This choice is motivated by the observation that the unstable pole of the open loop system, given by equation (5.8), lies at around 1.4 Hz.

Note that, due to its periodicity, the above random phase multisine signal prevents spectral leakage and allows strongly consistent estimates even in the feedback set up [16]. Also, due to its randomness, the impact of the nonlinearities can be estimated by comparing averages over consecutive periods and different realizations, see [17].

Figure 5.9 depicts the block diagram of the set up, where the excitation $r(t)$ is given by equation (5.27). A total number of $P = 10$ periods is recorded containing $R = 4$ different realizations. Let $Y(\omega)^{[r,p]}$ denote the Fourier transformation of the pth output period of $y(t)$ when the rth multisine realization is exciting the system. Analogously, $U(\omega)^{[r,p]}$ denotes the Fourier transform of the pth input period of $u(t)$ of the rth realization.

The averages $Y(\omega)^{[r]}$ and $U(\omega)^{[r]}$ over one realization are given by

$$Y(\omega)^{[r]} = \frac{1}{P} \sum_{p=1}^{P} Y(\omega)^{[r,p]} e^{-j\chi_\omega} \quad \text{and} \tag{5.28}$$

$$U(\omega)^{[r]} = \frac{1}{P} \sum_{p=1}^{P} U(\omega)^{[r,p]} e^{-j\chi_\omega}, \tag{5.29}$$

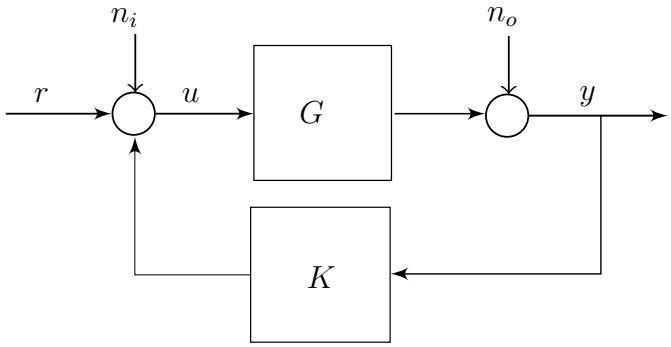


Figure 5.9: The block diagram of the system identification procedure. The Cubli (G) is controlled by a nominal linear feedback controller K and is excited by the random phase multisine signal r. n_i and n_0 denote the Gaussian white noise on the input and output of the system.

together with the sample covariances,

$$\hat{\sigma}^2_{XZ}(\omega)^{[r]} = \frac{1}{P(P-1)} \sum_{p=1}^{P} \left[(X(\omega)^{[r,p]} - X(\omega)^{[r]})(Z(\omega)^{[r,p]} - Z(\omega)^{[r]})^* \right], \tag{5.30}$$

where X and Z can be Y or U and * denotes the complex conjugate. Equations (5.28) and (5.29) account for the different phases of each multisine realization, allows the calculation of the means

$$Y(\omega) = \frac{1}{R} \sum_{r=1}^{R} Y(\omega)^{[r]}, \quad \text{and} \tag{5.31}$$

$$U(\omega) = \frac{1}{R} \sum_{r=1}^{R} U(\omega)^{[r]}. \tag{5.32}$$

Since the underlying system is nonlinear, different realizations of the excitation signal $r(t)$ will lead to different FRF estimates. Hence, by comparing the covariance over different realizations to the covariance within one realization, the impact of nonlinearities can be evaluated.

The uncertainty due to noise is computed by [17]

$$\hat{\sigma}^2_{XZ,n}(\omega) = \frac{1}{R^2} \sum_{r=1}^{R} \hat{\sigma}^2_{XZ}(\omega)^{[r]}$$

and can be compared to the total covariance

$$\hat{\sigma}^2_{XZ}(\omega) = \frac{1}{R(R-1)} \sum_{r=1}^{R} \left[(X(\omega)^{[r]} - X(\omega))(Z(\omega)^{[r]} - Z(\omega))^* \right]$$

in order to approximate the effect of nonlinearities.

The parameters η_{eb} are identified by minimizing the prediction error

$$e(\eta_{eb}, \omega) = Y(\omega) - G(\eta_{eb}, \omega)U(\omega), \tag{5.33}$$

weighted by the inverse of its covariance, namely

$$\begin{aligned} C_e(\eta_{eb}, \omega) = \hat{\sigma}^2_{YY}(\omega) + G(\eta_{eb}, \omega)\hat{\sigma}^2_{UU}(\omega)G(\eta_{eb}, \omega)^* \\ - \hat{\sigma}^2_{YU}(\omega)G(\eta_{eb}, \omega)^* - G(\eta_{eb}, \omega)\hat{\sigma}^2_{YU}(\omega)^*. \end{aligned} \tag{5.34}$$

This results in the sample maximum likelihood estimator [16] of the parameters η_{eb}, for which the cost function is given by

$$V_{SML}(\eta_{eb}) = \sum_{\forall \omega : A_\omega \neq 0} e(\eta_{eb}, \omega)^* C_e^{-1}(\eta_{eb}, \omega) e(\eta_{eb}, \omega). \tag{5.35}$$

The cost function $V_{SML}(\eta_{eb})$ is minimized using the Levenberg-Marquardt method. Compared to other optimization methods, only the computation of the gradient is required. Furthermore the gradient can be used to provide a covariance estimate of the parameters [18].

Figure 5.10 is an exemplary identification result of the transfer function from the input torque to the angular velocity $\dot{\varphi}$. Note that the residuals, i.e. the absolute differences between the parametric fit and the measurement result, are of the same order as the noise level, indicating a good fit.

Finally, the parametric model is validated by using an extra dataset containing 10 periods of 1 multisine realization. Figure 5.11 shows a section of one output period. Note that the predicted output, depicted in black, stays in between the two sigma bounds of the measured output and confirms the accuracy of the model on the validation set. The variance is approximated by the sample covariance over the 10 validation periods.

Corner Balancing

Repeating the identification on three perpendicular edges allows the assessment of the center of mass, the inertia tensor Θ_w and the diagonal elements of the inertia tensor $\hat{\Theta}_0$. These estimates can be used to design a feedback controller for stabilizing the Cubli on its corner.

In order to refine the estimates and to determine the off-diagonal entries of the inertia

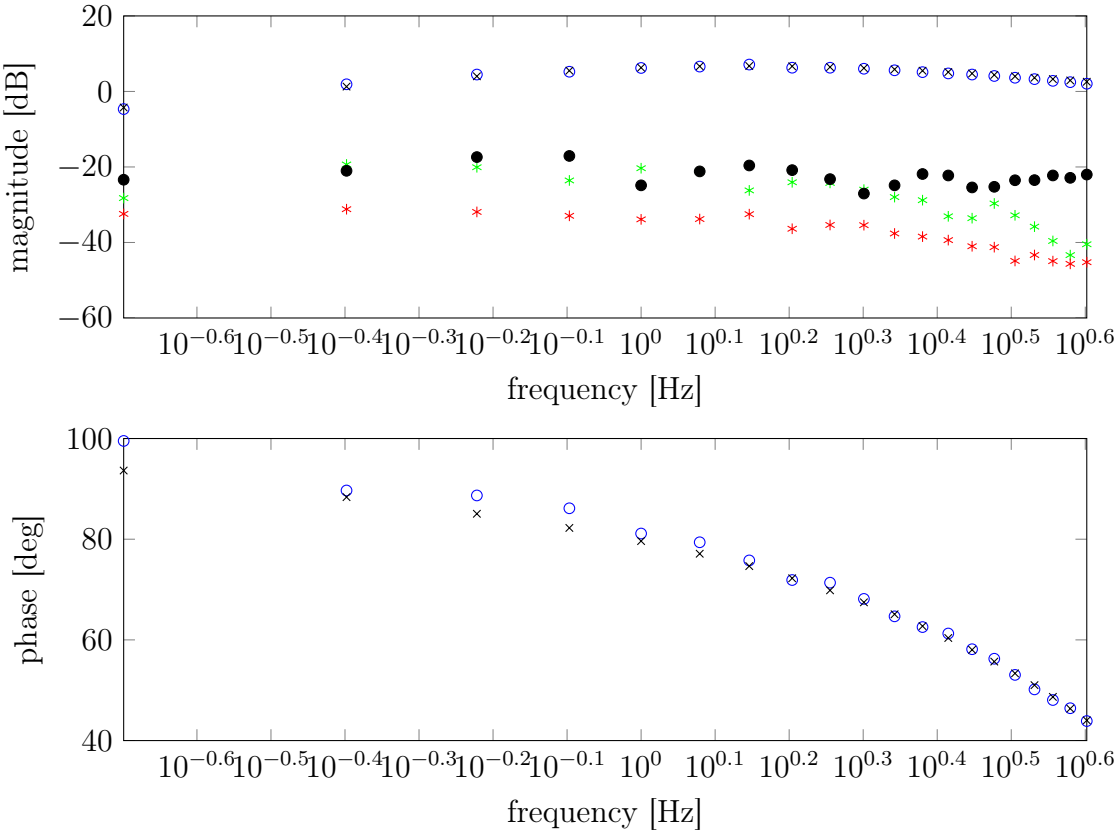


Figure 5.10: Frequency response related to the transfer function G_2, where the input represents the torque applied to the reaction wheel and the output represents the angular velocity $\dot{\varphi}$. The magnitude and phase of the measured transfer function is depicted in blue. The parametric fit is shown by black crosses, whereas the standard deviation of the transfer function measurement is depicted in light green. The absolute error between the parametric model and the measured response is indicated by black dots. Finally, the standard deviation due to the measurement noise is depicted by red stars.

tensor $\hat{\Theta}_0$, the identification procedure is repeated when the Cubli balances on its corner.

The approach parallels exactly the methodology delineated in the previous subsection and is therefore not presented. Because a full MIMO system must be identified in this case, more data is needed and minimizing the sample maximum likelihood cost function requires a greater computational effort. Nevertheless, it provides parameter estimates of around 1% accuracy (95% confidence interval) and contributes to a reasonable control performance, see Sec. 5.8.

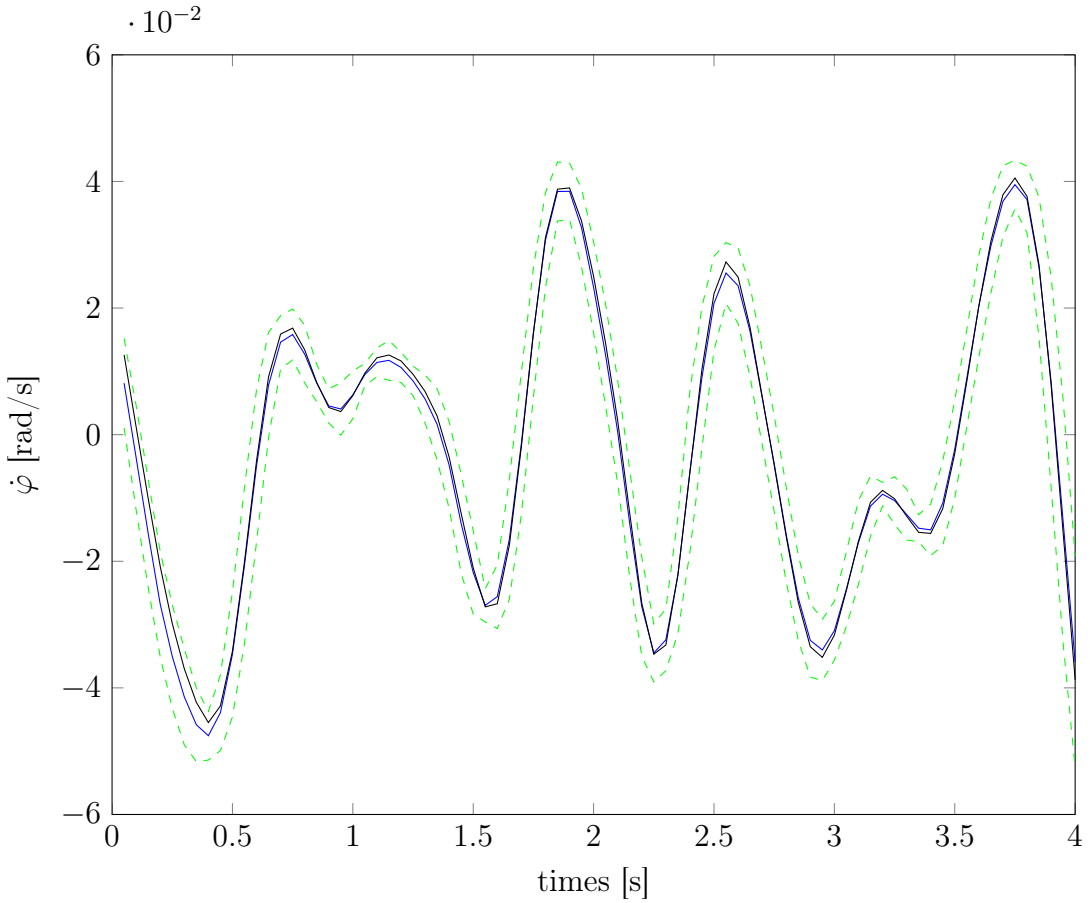


Figure 5.11: Validation in the time domain. Depicted in blue is the average of the angular velocity $\dot{\varphi}$ over 10 validation periods together with its 95% confidence interval. The output predicted by the parametric model is shown in black.

5.6 Balancing Control

Edge Balancing

The continuous time dynamics of the Cubli balancing on an edge is given in equations (5.8) and (5.9). A 20 ms sampling time was selected for the digital control, considering the open loop unstable pole of (5.8) (700 ms) and a safety factor of around 35. Using this sampling time the continuous time system was discretized using zero-order hold and the resulting discrete time model is given by

$$x[k+1] = A_d x[k] + B_d u[k], \quad y[k] = Cx[k] \quad k \in \mathbb{N}_0, \tag{5.36}$$

where A_d and B_d are the discrete-time counterparts of the continuous time state matrix A and input matrix B. For the sake of simplicity we use the same notation to represent the continuous and discrete time versions of the state x and input u.

Using the above discrete time model, a Linear Quadratic Regulator (LQR) feedback controller of the form

$$u[k] = -K_d y[k] \tag{5.37}$$

was designed, where $K_d \in \mathbb{R}^3$ is the LQR feedback gain that minimizes the cost function

$$J(u) := \sum_{k=1}^{\infty} x^{\mathrm{T}}[k]Qx[k] + u^{\mathrm{T}}[k]Ru[k], \quad Q \geqslant 0, R > 0.$$

A nonlinear edge balancing controller is presented in [14].

Corner Balancing

Conservation of Angular Momentum From (5.17) it follows that the rate of change of $\vec{p}_{\omega_h}$ lies always orthogonal to $\vec{m}$ and $\vec{g}$. Since $\vec{g}$ is constant, $\vec{p}_{\omega_h}$ will never change its component in direction $\vec{g}$ throughout the trajectory. Expressed in the Cubli's body fixed coordinate frame, it can be written as $\frac{d}{dt}(p_{\omega_h}^{\mathrm{T}} g) \equiv 0$, and this is nothing but the conservation of angular momentum around the axis $\vec{g}$.

This has a very important consequence for the control design: independent of the control input applied, the momentum around $\vec{g}$ is conserved and, depending on the initial condition, it may be impossible to bring the system to rest. For example, a yaw motion in the upright position can be slowed down by increasing the velocity of the reaction wheels. The yaw motion and the reaction wheel velocity can not both be driven to zero at the same time.

State Space Since the subsequent analysis and control will be carried out in the fixed body coordinate frame, the state space is defined by the set $\mathcal{X} = \{x = (g, p_{\omega_h}, p_{\omega_w}) \in \mathbb{R}^9 \mid ||g||_2 = 9.81\}$. Note that $\omega_h = \hat{\Theta}_0^{-1}(p_{\omega_h} - p_{\omega_w})$.

Equilibria As can be seen from (5.17), the condition $\dot{\vec{p}}_{\omega_h} = 0$ is fullfilled only if $m \parallel g$, i.e., $g = \pm\frac{m}{||m||_2}||g||_2$. Note that additional relative equilibria exist, but are not considered in the following. From (5.18) it follows that p_{ω_w} is constant and T is zero. Using (5.13) and (5.14) leads to $\omega_h = \hat{\Theta}_0^{-1}(p_{\omega_h} - p_{\omega_w}) \parallel g$ and $p_{\omega_h} \parallel g$, which corresponds exactly to the conserved part of p_{ω_h}. Note that $a \parallel b$ implies that the two vectors $a \in \mathbb{R}^3$ and $b \in \mathbb{R}^3$ are parallel. Combining everything together results in the following equilibria:

$$\begin{aligned}
\mathcal{E}_1 = \{(x,T) \in \mathcal{X} \times \mathbb{R}^3 \mid g^{\mathrm{T}} m = -||g||_2||m||_2, \\
p_{\omega_h} \parallel m,\ \hat{\Theta}_0^{-1}(p_{\omega_h} - p_{\omega_w}) \parallel m,\ T = 0\}, \\
\mathcal{E}_2 = \{(x,T) \in \mathcal{X} \times \mathbb{R}^3 \mid g^{\mathrm{T}} m = ||g||_2||m||_2, \\
p_{\omega_h} \parallel m,\ \hat{\Theta}_0^{-1}(p_{\omega_h} - p_{\omega_w}) \parallel m,\ T = 0\}.
\end{aligned}$$

Linearization reveals that the upright equilibria $\mathcal{E}_1$ is unstable, while the hanging equilibria $\mathcal{E}_2$ is stable in the Lyapunov [19] sense.

Nonlinear Control of the Reaction Wheel-based 3D Inverted Pendulum Let us first define the control objective. Since the angular momentum $\vec{p}_{\omega_h}$ is conserved in the direction of $\vec{g}$, the controller may only bring the component of $\vec{p}_{\omega_h}$ that is orthogonal to $\vec{g}$ to zero. Hence it is convenient to split the vector $\vec{p}_{\omega_h}$ into two parts: one in the direction of $\vec{g}$, and one orthogonal to it, i.e.

$$\vec{p}_{\omega_h} = \vec{p}_{\omega_h}^{\perp} + \vec{p}_{\omega_h}^{g} \quad \text{and} \quad \vec{p}_{\omega_h}^{g} = (\vec{p}_{\omega_h}^{\mathrm{T}}\vec{g})\frac{\vec{g}}{||\vec{g}||_2^2}.$$

From (5.17) and the conservation of angular momentum, it follows directly that

$$^{B}(\dot{\vec{p}}_{\omega_h}^{\perp}) = \dot{p}_{\omega_h}^{\perp} + \tilde{\omega}_h p_{\omega_h}^{\perp} = \tilde{m}g. \tag{5.38}$$

Another reasonable addition to the control objective is the asymptotic convergence of the angular velocity of the Cubli, ω_h, to zero. Consequently, the control objective can be formulated as driving the system to the closed invariant set $\mathcal{T} = \{x \in \mathcal{X} \mid g^{\mathrm{T}}m = -||g||_2||m||_2,\ \omega_h = \hat{\Theta}_0^{-1}(p_{\omega_h} - p_{\omega_w}) = 0,\ p_{\omega_h}^{\perp} = 0\}$.

In order to prove asymptotic stability, the hanging equilibrium must be excluded (as will become clear later). This can be done by introducing the set $\mathcal{X}^- = \mathcal{X} \setminus x^-$, where x^- denotes the hanging equilibrium with $\omega_h = 0$:

$$x^- = \{x \in X \mid g = \frac{||g||_2}{||m||_2}m,\ p_{\omega_h}^{\perp} = 0,\ p_{\omega_h} = p_{\omega_w}\}.$$

Next, consider the controller

$$u = K_1\tilde{m}g + K_2\omega_h + K_3 p_{\omega_h} - K_4 p_{\omega_w}, \tag{5.39}$$

where

$$\begin{aligned}
K_1 =& (1 + \beta\gamma + \delta)I + \alpha\hat{\Theta}_0,\\
K_2 =& \alpha\hat{\Theta}_0\tilde{p}_{\omega_h}^{\perp} + \beta\tilde{m}\tilde{g} + \tilde{p}_{\omega_h},\\
K_3 =& \gamma(I + \alpha\hat{\Theta}_0(I - \frac{gg^{\mathrm{T}}}{||g||_2^2})),\\
K_4 =& \gamma I, \quad \alpha, \beta, \gamma, \delta > 0,
\end{aligned}$$

and $I \in \mathbb{R}^{3\times 3}$ is the identity matrix.

THEOREM 5.1 The controller given in (5.39) makes the closed invariant set $\mathcal{T}$ of the system defined by (5.16)-(5.18) stable and asymptotically stable on $x \in \mathcal{X}^{-}$. ■

Proof. See [14]. □

5.7 Jump-Up

The *jump-up* can refer to either: 1) the movement from the lying flat position to the balancing on an edge position (face-to-edge jump); or 2) the movement from the balancing on an edge position to the balancing on a corner position (edge-to-corner jump). The huge amount of torque required to achieve these motions is produced by rapidly braking the reaction wheels, which are rotating at high angular velocities. More details on the braking mechanism can be found in Sec. 5.2 and Fig. 5.4.

Figure 5.12 shows the velocity profile of a reaction wheel during the rapid braking phase, where the braking command is sent to the RC servo at $t = 0$. The fixed time delay between sending the command and the brake pads touching the wheel can be acquired by observing a velocity profile as in Fig. 5.12. Once the brake pads start to squeeze in, after a brief transient phase (20 ms), an almost constant torque is applied to the reaction wheel.

Face-to-Edge Jump

In order to perform a face-to-edge jump, the reaction wheel with its axis parallel to the edge is accelerated to a desired angular velocity using a PI controller, and then rapidly braked. Due to the limitations in the motor torques that are used for balancing (low recovery angles), the jump is required to bring the Cubli sufficiently close to equilibrium state. However, the first principle model that is used to calculate $\dot{\psi}_0$, the desired angular velocity of the reaction wheel before braking, is not accurate and fails to give an appropriate value for the angular velocity. In order to solve this issue, this subsection presents a learning strategy that learns the appropriate angular velocity $\dot{\psi}_0$ with multiple trials, assuming that the Cubli as a system is time invariant.

The learning strategy is based on evaluating the difference in the total energy of the Cubli at the equilibrium $(\varphi, \dot{\varphi}) = (0, 0)$ and at a position φ_c that is closest to the equilibrium given by

$$\Delta E = \frac{1}{2}\hat{\Theta}_0 \dot{\varphi}_c^2 - mg\left(1 - \cos\varphi_c\right), \tag{5.40}$$

where $\dot{\varphi}_c$ is the angular velocity of the Cubli at $\varphi = \varphi_c$. Using ΔE, $\dot{\psi}_0$, the angular velocity of the reaction wheel before braking, is adjusted by $\Delta\dot{\psi}_0 = -\operatorname{sgn}(\dot{\psi}_0)\ k_E \cdot \Delta E$.

The total energy at any position after the brakes are released remains constant, since no further input torque is applied. Picking the position closest to the equilibrium allows the formulation of another evaluation criterion for the jump-up in the edge-to-corner case that is discussed later.

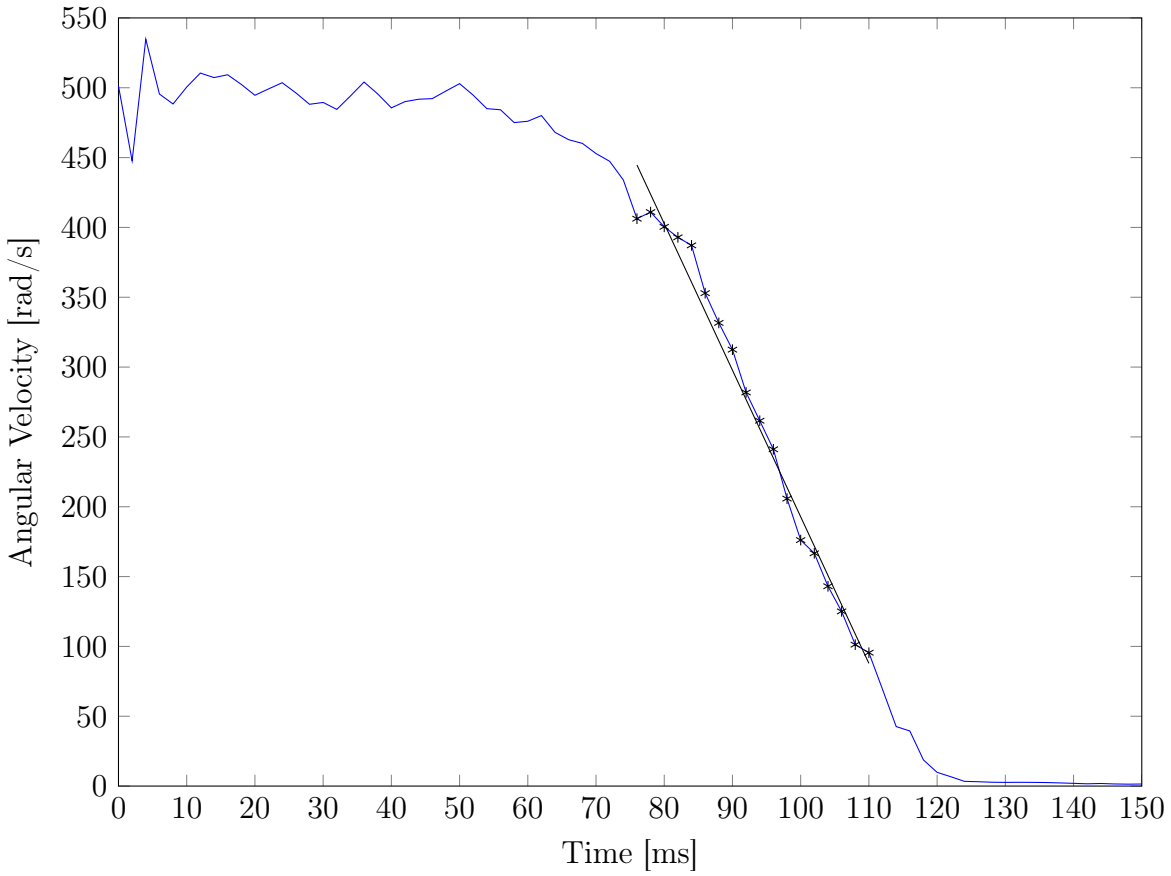


Figure 5.12: Braking profile.

Assume that the braking torque remains constant after the brief transient phase as shown in Fig. 5.12. This assumption gives $\Delta E \approx \dot{\varphi}_0 \Theta_w \Delta\dot{\psi}_0$, where $\dot{\varphi}_0$ is the angular velocity of the Cubli right after brakes are released. This fact can now be used to determine an appropriate value for k_E, giving $k_E = \frac{1}{\Theta_w \dot{\varphi}_0}$.

Catching Controller In addition to finding the appropriate angular velocity $\dot{\psi}_0$, once the brakes are released, the motor torques can be used to *guide* the Cubli along an ideal trajectory that will take it to the equilibrium. One ideal trajectory would be the one that takes the Cubli to the equilibrium position with out any control input. Since the absence of input will conserve energy, the above ideal trajectory $(\varphi^*, \dot{\varphi}^*)$ satisfies

$$\frac{1}{2}\hat{\Theta}_0 \dot{\varphi}^{*2} - mg\,(1 - \cos\varphi^*) = 0. \tag{5.41}$$

Now using the approximation $\cos\varphi \approx 1 - \varphi^2/2$ gives $\dot{\varphi}^* \approx -\sqrt{\frac{mg}{\hat{\Theta}_0}} \cdot \varphi^*$; this can be used to design the following linear state feedback controller, which can be used for both guiding

the Cubli and making it balance:

$$T = k_1 \cdot \left(\varphi + \sqrt{\frac{\hat{\Theta}_0}{mg}} \cdot \dot{\varphi} \right) + k_2 \cdot \dot{\psi}. \tag{5.42}$$

The above controller will not take any action while following the ideal trajectory, and will bring the Cubli towards the ideal trajectory if perturbed. The controller gains k_1 and k_2 can be tuned using pole placement for instance. Note that, for the sake of simplicity and because the value is negligible in practice, the above energy-based analsyis ignores the energy of the reaction wheels caused by their relative motion to the frame after braking.

Edge-to-Corner Jump

Assume, without loss of generality, that the Cubli is balancing on the edge coinciding with the ${}_B e_2$ axis as shown in Fig. 5.13. In order to jump up to the corner balancing position, a torque in the direction of ${}_I e_1$ is required. This torque can only achieved by a combined rapid braking of the reaction wheels with axes parallel to ${}_B e_1$ and ${}_B e_3$. For the same reasons as in the face-to-edge jump-up, the first principle model fails to provide appropriate angular velocities for the reaction wheels.

This subsection proposes a learning algorithm that learns the appropriate angular velocities of both reaction wheels through repeated trials. At every iteration, the quality of the jump-up trajectory, see Fig. 5.14, is quantified using the following two measures and, based on these measures, the wheel velocities are adjusted accordingly.

- Energy Measure: Similar to the face-to-edge jump, this measure focuses on the energy difference ΔE between Cubli's equilibrium position and the position that is closest to the equilibrium, denoted by c in Fig. 5.14. Note that the component of the Cubli's angular velocity parallel to m is not considered in the energy measure since it has almost no influence in taking the Cubli up towards the equilibrium.

- Divergence Measure: An ideal edge-to-corner jump-up trajectory will lie on the plane that contains $\vec{m}_0$ and $\vec{g}$, see Fig 5.14. The divergence measure quantifies the deviation of the trajectory from this plane with the following quantity:

$$\begin{aligned} \Delta D &= \left(\dot{\vec{m}} \times \vec{\delta c} \right)^{\mathrm{T}} \frac{\vec{m}_c}{\parallel m \parallel}, \\ &= \left((\vec{\omega}_h \times \vec{m}) \times \vec{\delta c} \right)^{\mathrm{T}} \frac{\vec{m}_c}{\parallel m \parallel}, \end{aligned}$$

  where $\vec{\delta c} = -\vec{g} / \parallel g \parallel - \vec{m} / \parallel m \parallel$.

Using the above two measures, the angular velocity of the two reaction wheels $\omega_{w0}(i)$

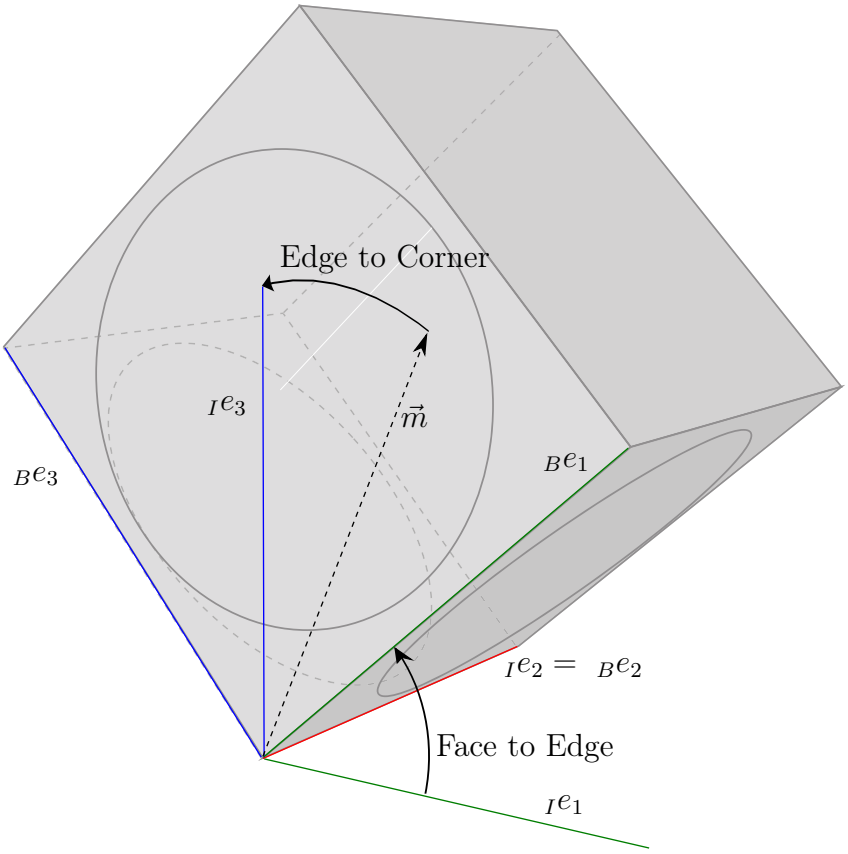


Figure 5.13: The Cubli is balancing on the edge parallel to the ${}_B e_2$ axis. In order to jump up to the corner balancing position, a torque in the direction of ${}_I e_1$ is required. This torque can only be achieved by a combined rapid braking of the reaction wheels with axes parallel to ${}_B e_1$ and ${}_B e_3$.

$i = 1, 3$ are updated with

$$\Delta\omega_{w0}(i) = -\operatorname{sgn}(\omega_w(i))\ k_E \Delta E - k_D \Delta D, \tag{5.43}$$

where $k_E \approx \frac{1}{\Theta_w \cdot (\omega_{b0}(1) + \omega_{b0}(3))}$, k_D is huristicly chosen constant, and $\omega_{b0}(i)$ $i = 1, 3$ is the ith component of ω_b right after the brakes are released. Figure 5.15 shows the performance of the above learning algorithm, when the system started from different initial wheel velocity pairs that can not make the edge-to-corner jump.

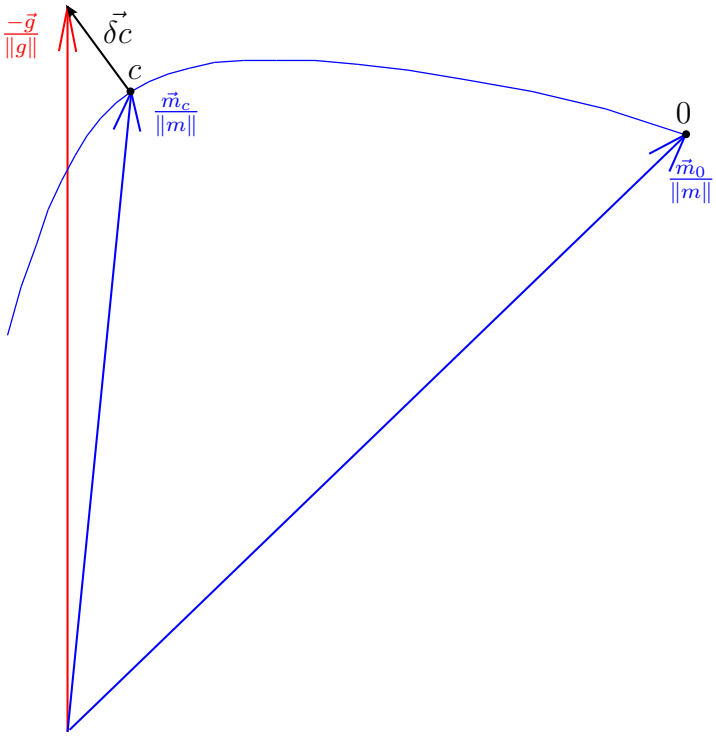


Figure 5.14: An exemplary trajectory (blue) of an edge-to-corner jump. The trajectory is defined by the motion of the normalized m vector. The point 0 denotes the position when the Cubli was balancing on an edge, and the point c denotes the position that is closest to the tip of the normalized g, the gravity vector.

5.8 Experimental Results

Balancing

This subsection presents disturbance rejection measurements of the Cubli balancing on its corner. Disturbance rejection measurements for the edge balancing case can be found in [20]. The controller given in (5.39) is implemented on the Cubli with a sampling time of $T_s = 20$ ms along with the algorithm proposed in [15] for state estimation.

Figures 5.16, 5.17, and 5.18 show disturbance rejection plots of the closed loop system with the proposed controller. A disturbance of roughly 0.1 Nm was simultaneously applied for 60 ms on each of the reaction wheels simultaneously. The control input is shown in Figure 5.19, where the controller reaches the steady state control input in less than 1 s. Finally, the controller attained a root mean square (RMS) inclination error of less than 0.025 ° at steady state.

Jump-Up

This subsection presents the evolution of the Cubli's states during a jump-up, where it first jumped up lying flat on its face to its edge and then it went from its edge to its

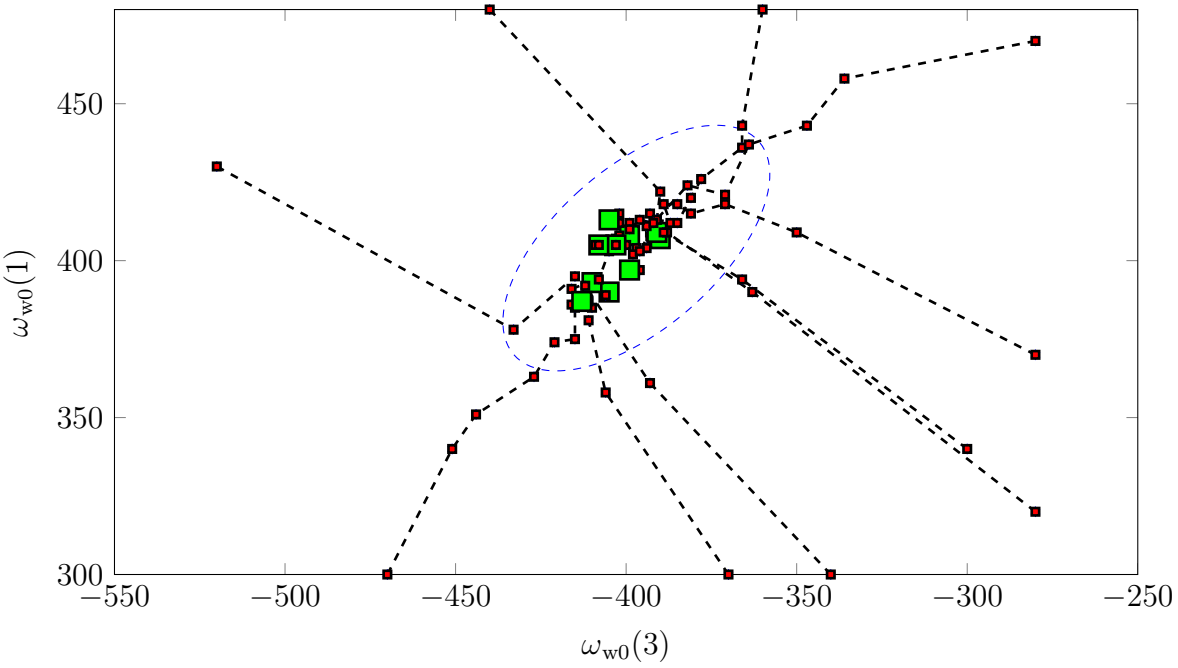


Figure 5.15: Learning the edge-to-corner jump. Each trajectory (black dashed line) shows the evolution of reaction wheel velocities $\omega_{w0}(1)$ and $\omega_{w0}(3)$ starting from different initial velocity pairs that can not make the edge-to-corner jump. Green squares show the converged values for the different initial velocity pairs. The experimentally found region inside the blue dotted ellipse contains all the velocity pairs that can make the edge-to-corner jump.

corner. Figure 5.20 shows the evolution of the Cubli's orientation in Euler angles and Fig. 5.21 shows the evolution of the reaction wheel velocities.

5.9 Conclusions and Future Work

This paper presented the design and implementation of, what is to date, the smallest 3D inverted pendulum and the first that can self erect. The paper started off with explaining the compact design that is based on off-the-shelf motors, motor controllers, brake-pads, etc. Then, the system was modelled using Lagrangian formalism. Introducing the concept of generalized momenta showed a clear relationship between the 1D and the 3D version. After the modelling was explained, a system identification procedure that does not require any external setup was described. Then a novel, back-stepping-based, nonlinear controller was proposed, before presenting the jump-up strategy along with a heuristic tuning method. Finally the results from several experiments were shown.

Future work includes, a rigorous tuning method for the nonlinear controller and the design of controllers that guide the Cubli along an *optimal* trajectory during the edge-to-corner jump-up.

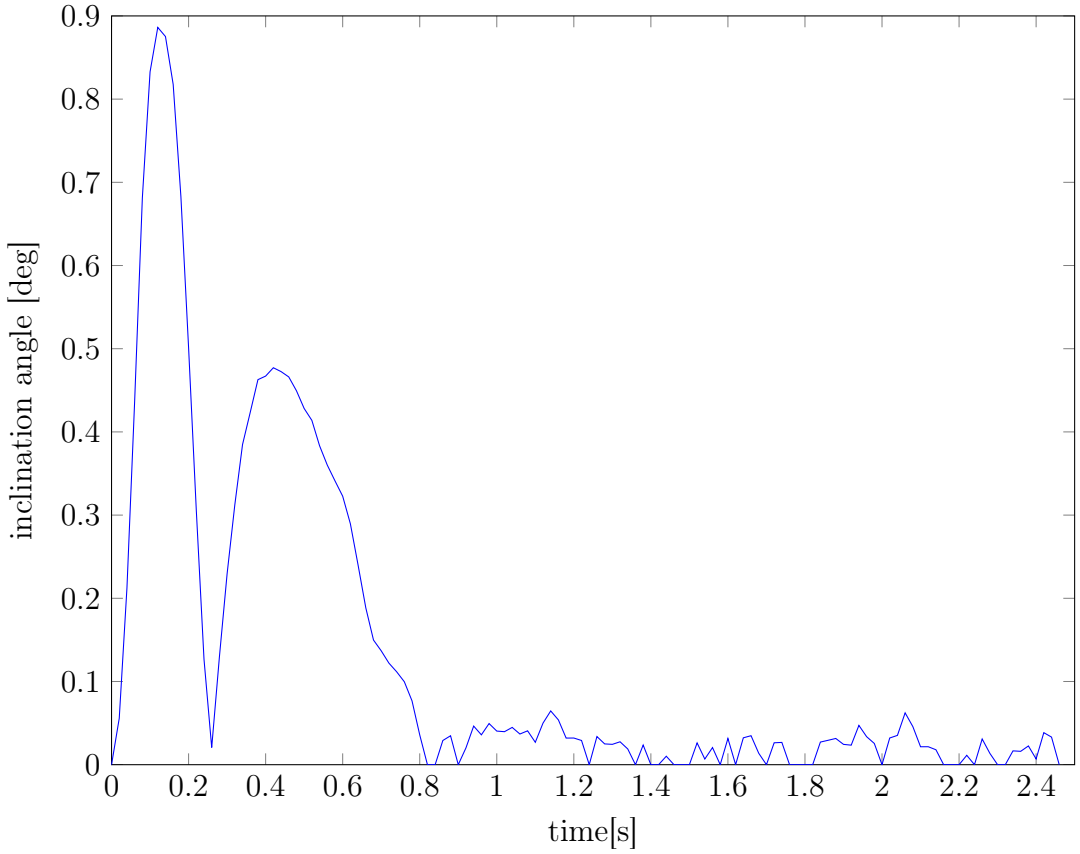


Figure 5.16: Disturbance rejection measurements. Inclination angle of the closed loop system versus time when approximately 0.1 Nm of torque was applied simultaneously on two reaction wheels for 60 ms as a disturbance.

5.10 Acknowledgements

The authors would like to express their gratitude towards Igor Thommen, Marc-Andrea Corzillius, Hans Ulrich Honegger, Alex Braun, Tobias Widmer, Felix Berkenkamp, Sonja Segmueller, and Michael Merz for their significant contribution to the mechanical and electronic design of the Cubli. We also thank Carolina Flores, Christine Pachinger, and Markus Waibel for their great help with graphics, media, and proof reading.

References

[1] A. Stephenson, "On a new type of dynamical stability," *Memoirs and Proceedings of the Manchester Literary and Philosophical Society*, vol. 52, no. 8, pp. 1–10, 1908.

[2] P. Reist and R. Tedrake, "Simulation-based LQR-trees with input and state constraints," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 5504–5510.

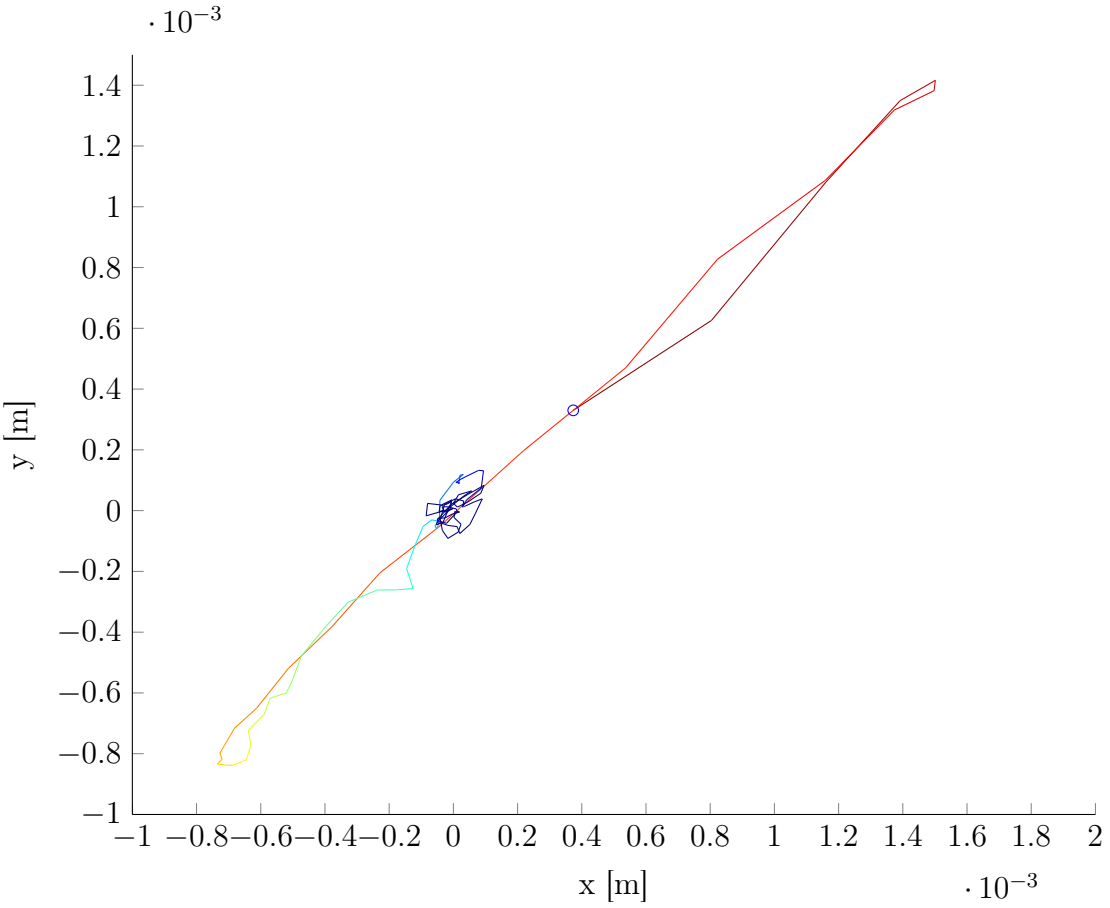


Figure 5.17: Disturbance rejection measurements. Trajectory of the center of mass projected onto the x-y plane in the inertial coordinate frame. The color gradient indicates the evolution over time. The blue circle highlights the starting point of the trajectory ($t = 80$ ms), right after the disturbance.

[3] D. Alonso, E. Paolini, and J. Moiola, "Controlling an inverted pendulum with bounded controls," in *Dynamics, Bifurcations, and Control*, ser. Lecture Notes in Control and Information Sciences, F. Colonius and L. Grüne, Eds. Springer Berlin Heidelberg, 2002, vol. 273, pp. 3–16.

[4] M. V. Bartuccelli, G. Gentile, and K. V. Georgiou, "On the stability of the upside-down pendulum with damping," *Proceedings: Mathematical, Physical and Engineering Sciences*, pp. 255–269, 2002.

[5] J. Meyer, N. Delson, and R. de Callafon, "Design, modeling and stabilization of a moment exchange based inverted pendulum," in *15h IFAC Symposium on System Identification, Saint-Malo, France*, 2009, pp. 462–467.

[6] D. Bernstein, N. McClamroch, and A. Bloch, "Development of air spindle and triaxial air bearing testbeds for spacecraft dynamics and control experiments," in *American Control Conference*, 2001, pp. 3967–3972.

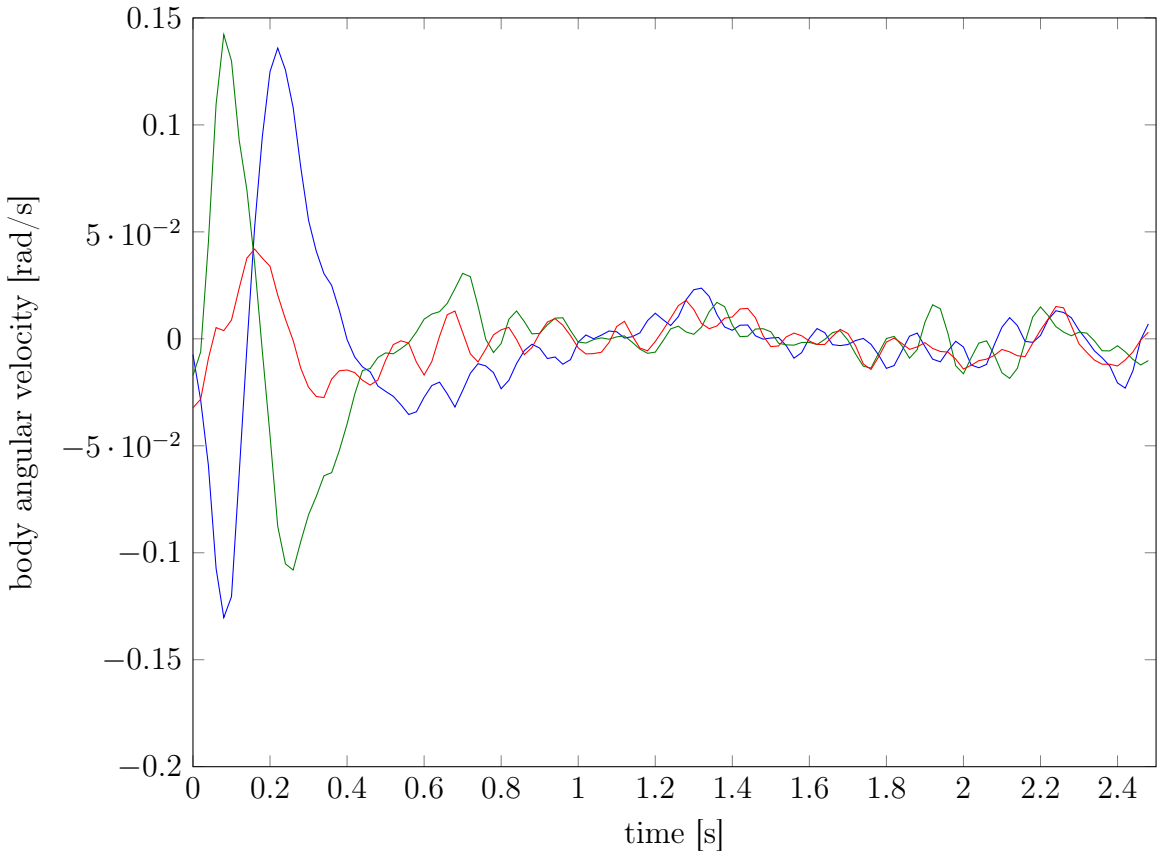


Figure 5.18: Disturbance rejection measurements. Angular velocity of the housing ω_h versus time. The different colors depict the different vector components of ω_h. Here, approximately 0.1 Nm of torque was applied simultaneously on two reaction wheels for 60 ms as a disturbance.

[7] S. Trimpe and R. D'Andrea, "The balancing cube: A dynamic sculpture as test bed for distributed estimation and control," *Control Systems, IEEE*, vol. 32, no. 6, pp. 48–75, Dec 2012.

[8] T. Yoshimitsu, T. Kubota, I. Nakatani, T. Adachi, and H. Saito, "Micro-hopping robot for asteroid exploration," *Acta Astronautica*, vol. 52, no. 2–6, pp. 441 – 446, 2003, selected Proceedings of the 4th {IAA} International conference on Low Cost Planetary Missions.

[9] M. Pavone, J. Castillo-Rogez, I. Nesnas, J. Hoffman, and N. Strange, "Spacecraft/rover hybrids for the exploration of small solar system bodies," in *2013 IEEE Aerospace Conference*, March 2013, pp. 1–11.

[10] R. Jones, "The muses cn rover and asteroid exploration mission," in *Proc. 22nd International Symposium on Space Technology and Science*, 2000, pp. 2403–2410.

[11] J. Romanishin, K. Gilpin, and D. Rus, "M-blocks: Momentum-driven, magnetic

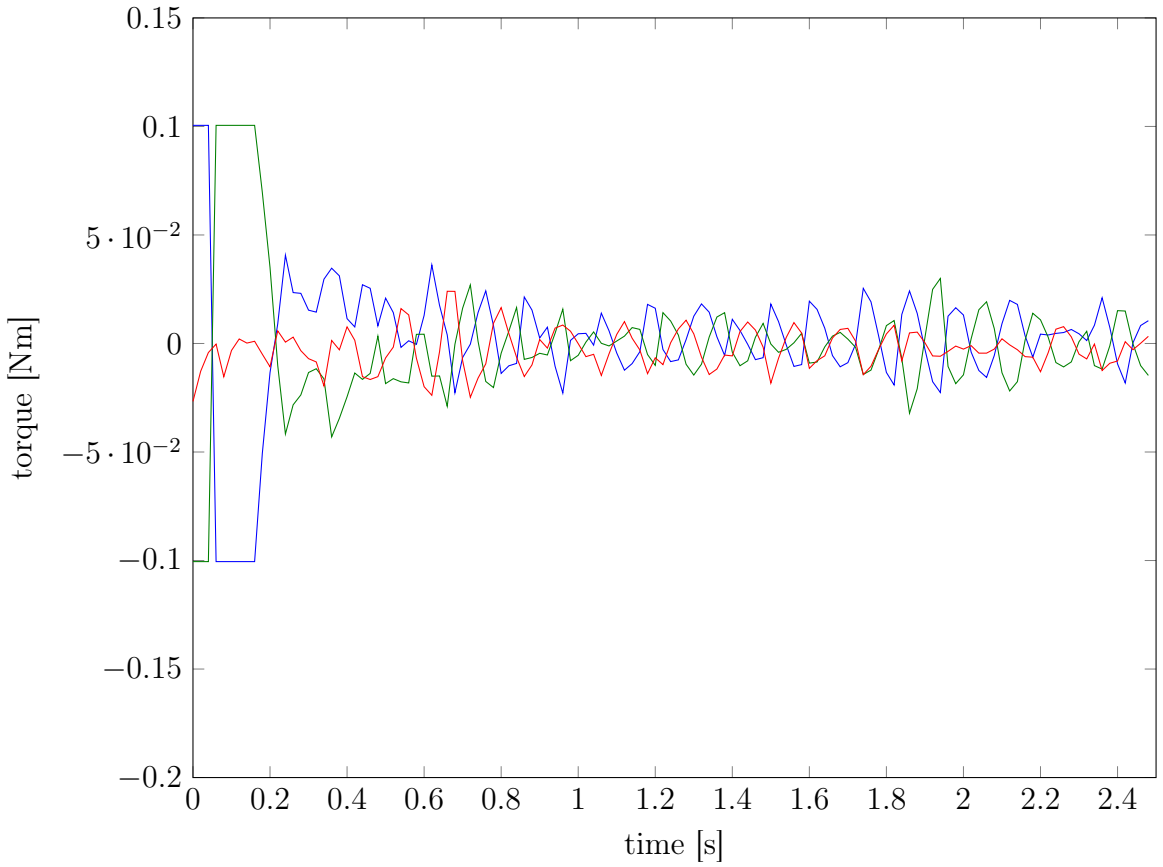


Figure 5.19: Disturbance rejection measurements. Depicted is the resulting control input of the nonlinear controller. The different colors depict the different vector components of T. Here, approximately 0.1 Nm of torque was applied simultaneously on two reaction wheels for 60 ms as a disturbance.

modular robots," in *Proc. 2013 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, Nov 2013, pp. 4288–4295.

[12] L. Cornelius, *The variational principles of mechanics*, 4th ed. University of Toronto Press Toronto, 1970.

[13] M. Gajamohan, M. Muehlebach, T. Widmer, and R. D'Andrea, "The cubli: A reaction wheel based 3d inverted pendulum," in *proc. European Control Conference*, 2013, pp. 268–274.

[14] M. Muehlebach, G. Mohanarajah, and R. D'Andrea, "Nonlinear analysis and control of a reaction wheel-based 3D inverted pendulum," in *Proc. IEEE Conference on Decision and Control (CDC)*, Florence, Italy, 2013, pp. 1283–1288.

[15] S. Trimpe and R. D'Andrea, "Accelerometer-based tilt estimation of a rigid body with only rotational degrees of freedom," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 2630–2636.

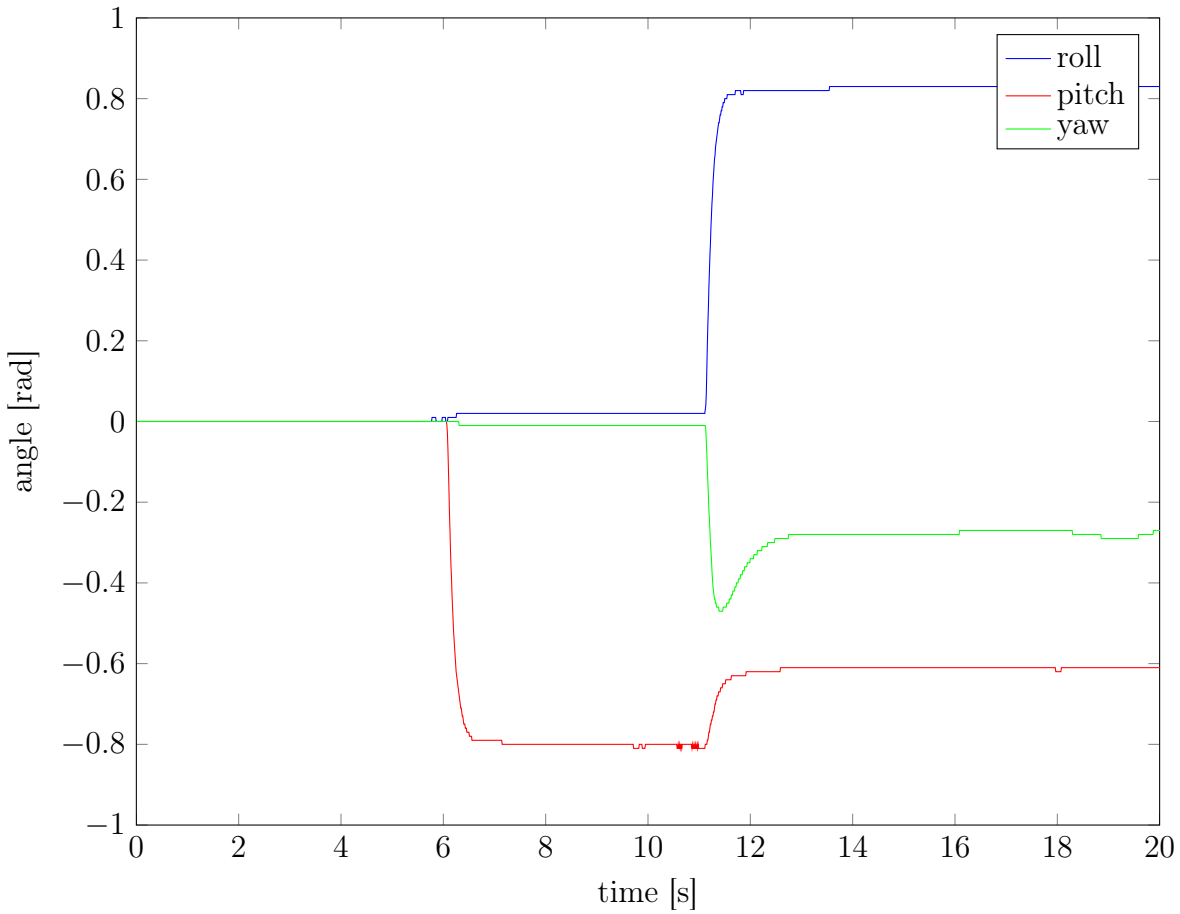


Figure 5.20: Evolution of the Cubli's orientation in Euler angles during the face-to-edge jump ($t \sim 6$ s) and edge-to-corner jump ($t \sim 11$ s).

[16] R. Pintelon and J. Schoukens, *System identification: a frequency domain approach*. John Wiley & Sons, 2004, ch. 8, pp. 285–287,301.

[17] ——, *System identification: a frequency domain approach*. John Wiley & Sons, 2004, ch. 3, pp. 86–88.

[18] ——, *System identification: a frequency domain approach*. John Wiley & Sons, 2004, ch. 7, pp. 212–217.

[19] H. Khalil, *Nonlinear Systems*. Upper Saddle River, New Jersey: Prentice Hall, 1996.

[20] M. Gajamohan, M. Merz, I. Thommen, and R. D'Andrea, "The Cubli: A cube that can jump up and balance," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 3722–3727.

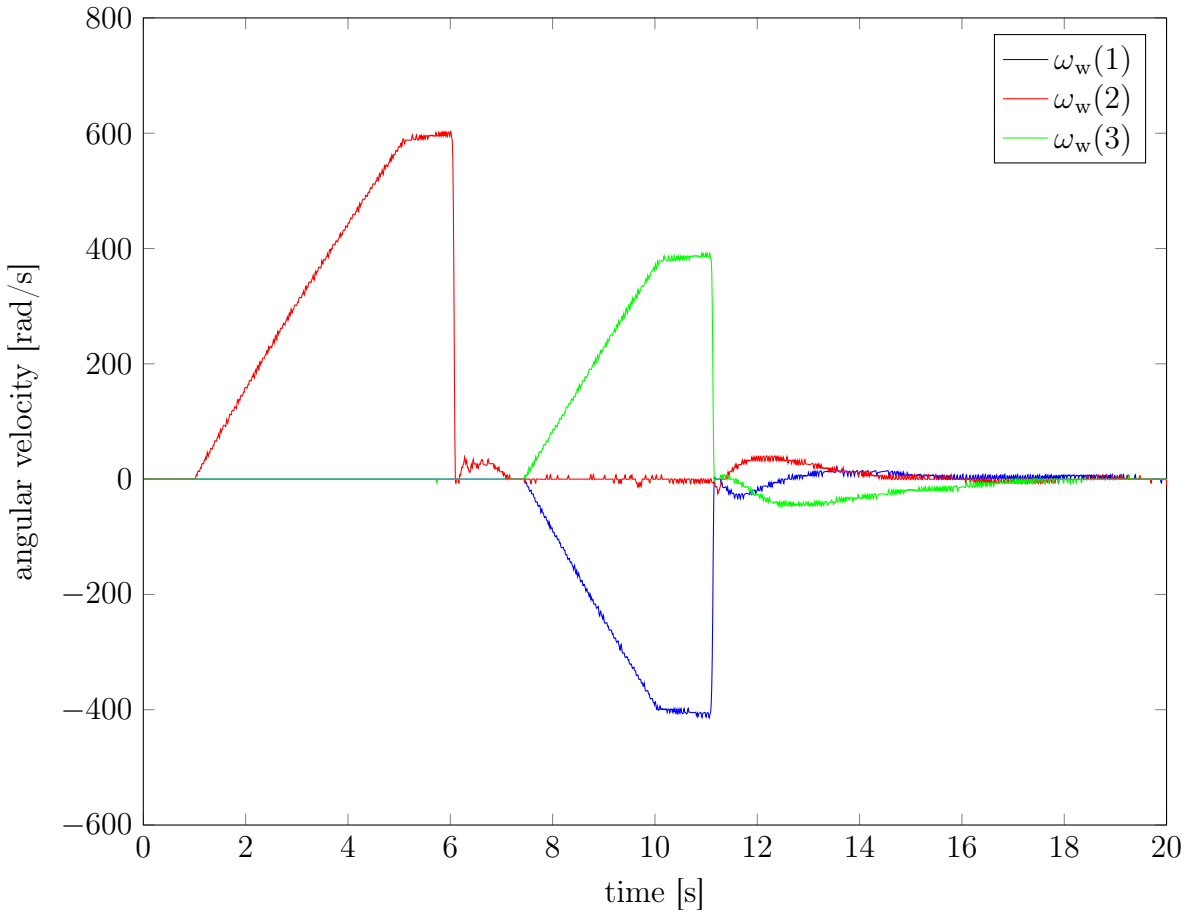


Figure 5.21: Evolution of the Cubli's reaction wheel velocities during the face-to-edge jump ($t \sim 6$ s) and edge-to-corner jump ($t \sim 11$ s).

6

Conclusions and Future Directions

This dissertation described the work on:

- *The cloud*: the design and implementation of a cloud robotics platform and a demonstrator for cloud-based collaborative 3D mapping
- *Paper planes*: algorithms to aggregate and learn from data gathered by various robots connected to the cloud
- *The cube*: a reaction wheel-based 3D inverted pendulum with a small foot print that can self erect

Conclusions that summarize the specific contributions and/or provide direct extensions to the work are presented at the end of Chapters 2-5. The following presents a retrospective of the work accomplished, some of the lessons learned, and future directions.

The Cloud

Major contributions of this work include:

- Design and implementation of a novel robotics-specific Platform-as-a-Service framework that helps robots to offload some of their heavy computation and acts as a communication medium between robots.
- Cloud-based collaborative 3D mapping in real-time: The developed system uses half the bandwidth of other comparable systems [1], and to date has the lowest cost (USD 500 per robot).

The biggest challenge in building the Rapyuta cloud robotics platform was in satisfying all the design constraints: simplicity for the end-user, good reactivity, extensibility, and compatibility with existing robotics software. Rapyuta is currently in its fourth design iteration, each of which involved substantial redesigns even though technology choices (such as WebSockets, Twisted asynchronous programming, and Linux containers)

remained largely unchanged. Each iteration introduced new functionalities to enhance the usability, security, and extensibility of the system. Because cloud robotics is still a relatively young and untested field, generating interest in and convincing roboticists of the advantages of such a system was another significant challenge. A concrete collaborative mapping demonstrator was built using low-cost robots to help overcome this problem (see Chapter 3).

The main future challenges include:

- *Semantics*: Although ROS provides standard message types for the information exchange between robotic software components, there is no semantic layer that gives *meaning* and *grounding* to a commonly agreed upon corpus such as WordNet [2] or KnowRob [3]. Having additional semantic information for messages and software components will allow software agents to perform reasoning on the configuration and pave the way towards fully autonomous systems.
- *Cloud-based configuration/control*: Currently, Rapyuta can only control the applications running in the cloud. Having the ability to configure and launch applications on the robot as well will facilitate highly configurable robotics systems. The challenge here lies in providing sufficient security such that control of the robot cannot be taken over by an unauthorized access.
- *Wireless connectivity*: Nowadays, robust high-bandwidth wireless connectivity is in general not an issue indoors due to ample number of access points provided by the infrastructure. However, outdoor connectivity remains an open question. Mobile broadband technologies such as LTE show promise in solving some of the issues. Additionally, mobile adaptive networks [4] - where access points are mounted on mobile robots that move around to provide the necessary bandwidth and robustness - also show promise for solving the outdoor connectivity issue.

Rapyuta will continue its development as an open source community project, with more than 10 active research groups using the platform and providing valuable feedback. The open source effort will be partially supported by the Amazon Web Services grant.

Paper Planes

Major contributions of this work include:

- Proposed a reinforcement learning algorithm, proved its convergence for trajectory tracking, and showed that a significant amount of knowledge can be transferred even between cases where the reference trajectories are not the same.
- Developed and implemented an Indirect Object Search algorithm that predicts occurrence and location probabilities of small and hard-to-detect object classes based on the occurrence and location of large, easy-to-detect object classes.

This work included the development of algorithms that aggregate and learn from the data gathered from various robots. High-level learning algorithms such as the Indirect

Object Search were implemented with relative ease compared to the hardware-dependent low-level algorithms such as Iterative Learning Control (ILC) or articulation model learning. The biggest challenge with respect to the low-level algorithms was to make them work across multiple hardware platforms and their various low-level software ecosystems.

The algorithmic work done under this dissertation just scrapes the surface of the realm of open issues. Learning should happen at various levels, starting from trajectories to semantic reasoning. Furthermore, learning algorithms can be used in the following ways to facilitate an internet for robots:

- *Imitation learning*: Learning a task by observing a teacher, typically a human, doing that specific task. These methods will help bootstrap the RoboEarth knowledge repository with useful high quality knowledge.
- *Learning from the World Wide Web*: Learning from pre-existing web-based information. Since the knowledge was originally meant to be used by humans, in order to use this information on a robot, the learning algorithm must augment the existing information with additional knowledge and remove uncertainties.
- *Anomaly detection*: Algorithms should be developed to detect anomalies in the data coming in from different robots in order to avoid misbehavior resulting from the use of accumulated data; such misbehavior could pose a threat to the robots and their surrounding environment.

The Cube

Major contributions of this work include:

- The smallest 3D inverted pendulum to date, and the first 3D pendulum that can self erect
- A novel nonlinear control design with intuitive tuning parameters

The Cubli is an ideal platform for research and education in nonlinear controls and state estimation. Application examples include: hybrid rovers for planetary exploration [5, 6] or self assembling robots [7].

The biggest challenge in terms of building the Cubli was the braking mechanism. Our initial proposal [8] used for the 1D prototype, shown in Fig. 6.1, was not suitable for the 3D version. Due to the high inertia/mass of the 3D prototype, the required angular velocities of the reaction wheels were very high. Instantaneously stopping these high speed wheels created a large impulsive force that caused the structure to deform and/or break. This issue was resolved by a complete redesign of the braking mechanism described in Chapter 5. Other features of this project include a nonlinear controller design with intuitive tuning parameters, and a system identification technique that does not require any additional apparatus.

For future work, multiple Cublis can be combined to build a testbed for studying simultaneous stabilization and synchronization in a distributed setting, see Fig 6.2. This

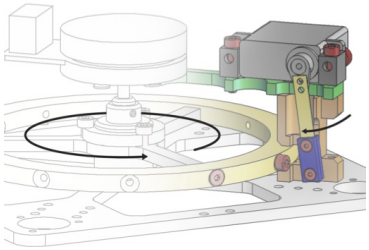


Figure 6.1: CAD drawing of the RC servo-based braking mechanism used at the project's outset: An RC servo is used to collide a metal barrier (blue) with the bolt head (red) attached to the momentum wheel. The sudden impact of the high speed wheel with the brake's metal barrier caused the structure to deform and/or break.

testbed can be conceptualized as a choreographed dance performance of Cublis that can balance autonomously on an edge or corner. The control system on each cube must achieve two simultaneous objectives: (1) stabilization of individual Cublis, and (2) motion synchronization of each Cubli with its peers through wireless communication links.

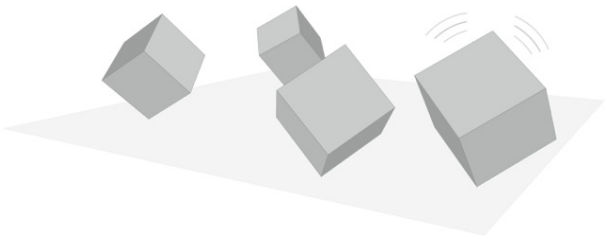


Figure 6.2: An illustration of multiple Cublis trying to synchronize their attitude and angular velocities while balancing on their corners.

References

[1] L. Riazuelo, J. Civera, and J. M. M. Montiel, "C2tam: A cloud framework for cooperative tracking and mapping," *Robotics and Autonomous Systems*, 2013, accepted for publication.

[2] G. A. Miller, "Wordnet: A lexical database for english," *COMMUNICATIONS OF THE ACM*, vol. 38, pp. 39–41, 1995.

[3] M. Tenorth and M. Beetz, "Knowrob — knowledge processing for autonomous personal robots," in *in IEEE/RSJ International Conference on Intelligent RObots and Systems*, 2009.

[4] S. Gil, D. Feldman, and D. Rus, "Communication coverage for independently moving robots," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, Oct 2012, pp. 4865–4872.

[5] M. Pavone, J. Castillo-Rogez, I. Nesnas, J. Hoffman, and N. Strange, "Spacecraft/rover hybrids for the exploration of small solar system bodies," in *Aerospace Conference, 2013 IEEE*, March 2013, pp. 1–11.

[6] T. Yoshimitsu, T. Kubota, I. Nakatani, T. Adachi, and H. Saito, "Micro-hopping robot for asteroid exploration," *Acta Astronautica*, vol. 52, no. 2–6, pp. 441 – 446, 2003, selected Proceedings of the 4th {IAA} International conference on L ow Cost Planetary Missions. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576502001868

[7] J. Romanishin, K. Gilpin, and D. Rus, "M-blocks: Momentum-driven, magnetic modular robots," in *Proc. 2013 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, Nov 2013, pp. 4288–4295.

[8] G. Mohanarajah, M. Merz, I. Thommen, and R. D'Andrea, "The Cubli: A cube that can jump up and balance," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura-Algarve, Portugal, 2012, pp. 3722–3727.

A

Appendix: Paper Planes

A.1 Proof of Proposition 4.3.1

To prove proposition 4.3.1 we start with the following lemma:

LEMMA A.1
$\forall \epsilon > 0 \; \exists \; \alpha_0(\epsilon, d) < \infty$ s.t. if $\omega_T = \mathcal{O}((\log T)^d)$, then $\omega_T \leq \alpha_0 T^{\epsilon}, \; T \geq 1.$ □

Proof. $\exists \; a < \infty$ s.t. $\omega_T \leq a(\log T)^d$. Take $\alpha_0 = a \max T^{-\epsilon}(\log T)^d$. Taking the derivative of $g(T) = T^{-\epsilon}(\log T)^d$:

$$\frac{d(\log T)^{d-1} - \epsilon(\log T)^d}{T^{1+\epsilon}} = 0 \tag{A.1}$$

The function $g(T)$ is continuous for $T \geq 1$ with $g(1) = 0$ and as $T \to \infty, \; g(T) \to 0$. Hence the only optimum attained in $(1, \infty)$ at $T_{opt} = \exp(d/\epsilon)$ with $g(T_{opt}) = \left(\frac{d}{e\epsilon}\right)^d$ is a global maximum. Taking $\alpha_0(\epsilon, d) = a\left(\frac{d}{e\epsilon}\right)^d$ then we can see that

$$\omega_T \leq a(\log T)^d \leq \alpha_0 T^{\epsilon}, \; T \geq 1. \tag{A.2}$$

□

Equipped with this lemma, we can prove Proposition 4.3.1 for the squared exponential kernel [1]:

[1] The proof is the same for the linear kernel, but the exponent of the bound (A.3) changes for Matérn kernels with $1 < \nu < \infty$.

Proof. We can bound (4.19) using the previous lemma:

$$\sum_{\ell=1}^{L(T)} \Lambda_\ell + \epsilon_s L \leq R_T \leq \sqrt{\kappa T \beta_T \gamma_T} \leq \alpha T^{3/4} \tag{A.3}$$

where $\omega_T^2 = \kappa\beta_T\gamma_T = 2B\kappa\gamma_T + 300\kappa\gamma_T^2 \log^3(T/\delta) = \mathcal{O}((\log T)^{2d+5})$ and $\alpha = \alpha_0(1/4, (2d+5)/2)$ for the squared exponential kernel. Since $\Lambda_\ell \geq 0,\ \ell = 1, \ldots, L$ we get w.h.p. $p \geq 1 - \delta$:

$$L \leq \frac{\alpha\ T^{3/4}}{\epsilon_s} \tag{A.4}$$

$$\tau_m = \frac{T}{L} \geq \frac{\epsilon_s\ T^{1/4}}{\alpha} \tag{A.5}$$

where τ_m denotes the average of the stopping times $\tau(\ell), \ell = 1, \ldots, L$. But this means $\exists \ell \leq L$ s.t. $\tau(\ell) \geq \tau_m \geq (\epsilon_s/\alpha)T^{1/4} \geq N$ before $T \leq T_{max} = \left(\frac{N\alpha}{\epsilon_s}\right)^4$. □

A.2 Numerical Examples

Table A.1: Quadrocopter dynamical constraints

Constraints	Values
f_{min}	0.25 m/s^2
f_{max}	5.5 m/s^2
$\dot{f}_{max}$	51 m/s^3
$\dot{\phi}_{max}$	25 rad/s
$\ddot{\phi}_{max}$	200 rad/s^2

Example 1. As an example consider the effect of wind on the quadrotor operation. Assuming the wind, coming at an angle of θ from the horizontal axis, exerts a pressure P_{wind} on the quadrotor with area A, the dynamics is modified as follows:

$$\begin{aligned} \ddot{y} &= -f_{\text{coll}} \sin\phi + P_{wind} A sin(\theta + \phi) cos\theta \\ \ddot{z} &= f_{\text{coll}} \cos\phi - g + P_{wind} A sin(\theta + \phi) sin\theta \\ \dot{\phi} &= \omega_x \end{aligned} \tag{A.6}$$

Mismatch in this case is only in the drift term. Using squared Euclidean distance and forward Euler integration with time discretization h the cost disturbance $\delta q_t(u; x, s) =$

$q_t - \hat{q}_t$ for the simple case of a perfectly horizontal wind, $\theta = 0$, can be calculated as follows:

$$\begin{aligned}\delta q_t(u; x, s) = (h^2 P_{wind}^2 A^2 - 2h^2 P_{wind} A \ f_{\text{coll}}) \sin^2 x(5) \\ + 2h P_{wind} A (x(2) - s(2)) \sin x(5)\end{aligned} \tag{A.7}$$

In this case we effectively learn to compensate for this repeating disturbance $\delta q_t(u; x, s) : \mathbb{R}^2 \times \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$, as we do online *TGP* optimization along the trajectory.

Example 2. As another example consider mismatch in the quadrotor actuators. If the actual applied force is $f_{\text{coll}}(1 + a)$ for some small unknown a the cost difference can be calculated as before:

$$\begin{aligned}\delta q_t(u; x, s) = (h^2 a^2 + 2ha) f_{\text{coll}}^2 \\ - 2h(x(2) - s(2)) a f_{\text{coll}} \sin x(5) \\ + 2h(x(4) - s(4) - g) a f_{\text{coll}} \cos x(5)\end{aligned} \tag{A.8}$$