On Robotic Work-Space Sensing and Control

Linderoth, Magnus

2013

Document Version:
Publisher's PDF, also known as Version of record

On Robotic Work-Space Sensing and Control

Magnus Linderoth

LUNDS UNIVERSITET

Department of Automatic Control

PhD Thesis
ISRN LUTFD2/TFRT--1098--SE
ISBN 978-91-7473-669-4 (print)
ISBN 978-91-7473-670-0 (web)
ISSN 0280–5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck.
Lund 2013

Abstract

Industrial robots are fast and accurate when working with known objects at precise locations in well-structured manufacturing environments, as done in the classical automation setting. In one sense, limited use of sensors leaves robots blind and numb, unaware of what is happening in their surroundings. Whereas equipping a system with sensors has the potential to add new functionality and increase the set of uncertainties a robot can handle, it is not as simple as that. Often it is difficult to interpret the measurements and use them to draw necessary conclusions about the state of the work space. For effective sensor-based control, it is necessary to both understand the sensor data and to know how to act on it, giving the robot perception–action capabilities.

This thesis presents research on how sensors and estimation techniques can be used in robot control. The suggested methods are theoretically analyzed and evaluated with a large focus on experimental verification in real-time settings.

One application class treated is the ability to react fast and accurately to events detected by vision, which is demonstrated by the realization of a ball-catching robot. A new approach is proposed for performing high-speed color-based image analysis that is robust to varying illumination conditions and motion blur. Furthermore, a method for object tracking is presented along with a novel way of Kalman-filter initialization that can handle initial-state estimates with infinite variance.

A second application class treated is robotic assembly using force control. A study of two assembly scenarios is presented, investigating the possibility of using force-controlled assembly in industrial robotics. Two new approaches for robotic contact-force estimation without any force sensor are presented and validated in assembly operations.

The treated topics represent some of the challenges in sensor-based robot control, and it is demonstrated how they can be used to extend the functionality of industrial robots.

Acknowledgments

First of all I would like to thank my supervisors Rolf Johansson and Anders Robertsson, who introduced me to the field of robotics. They complement each other well and have given me important support and guidance in the different aspects of my development as a researcher. I also want to thank Klas Nilsson, who in practice has acted as an extra supervisor. Much of my work has been done in close cooperation with Andreas Stolt, who has been a very good work partner. Coming up with solutions to problems and implementing them always works smoothly with Andreas. I also want to thank the rest of my colleagues in the ROSETTA project for many fruitful discussions and feedback on my work.

My interest for computer vision, which has been an important part of my thesis work, first started while following Kalle Åström's inspiring lectures. I have on numerous occasions discussed my image-analysis algorithms with Håkan Ardö, who has a large knowledge and insight about image analysis and implementation aspects. The meetings with Håkan have been interesting and fruitful, and afterwards I have always had lots of new ideas to try out.

I want to thank all my colleagues at the Department of Automatic Control for making it such a nice environment to work in. The coffee breaks and lunches often offered interesting discussions on all imaginable topics. In particular I want to thank Kristian Soltész, who played an important role when I decided to do a PhD and has provided valuable input to discussions on my research and many other topics.

Some people have been particularly important for making the work run smoothly. Anders Blomdell, with his vast programming experience, served as the living book of answers when I had programming problems, and the robots in the lab could not run without him. Blomdell has together with Leif Andersson and Anders Nilsson helped out with computer-related problems and made sure that we have a functioning computer environment. Rolf Braun and Pontus Andersson have made sure that the labs are in good shape. Eva Schildt, Britt-Marie Mårtensson, Agneta Tuszynski,

Eva Westin, Ingrid Nilsson, Monika Rasmusson, and Lizette Borgeram have helped me handle all administrative matters.

I want to thank my family for supporting me and laying the foundation that got me where I am today. Finally, I want to thank my lovely Kerstin for making me feel happy and giving me energy as well as giving me valuable feedback on my work while writing this thesis.

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007–2013 – Challenge 2 – Cognitive Systems, Interaction, Robotics – under grant agreement No 230902 - ROSETTA. This document reflects only the author's views and the European Community is not liable for any use that may be made of the information contained herein.

The author is a member of the LCCC Linnaeus Center and the eLLIIT Excellence Center at Lund University.

Contents

1. Introduction **11**
1.1 Motivation and Background 11
1.2 Publications 12
1.3 Outline and Contributions 14

Part I Vision-Based Control **17**
2. Ball-Catching Robot **19**
2.1 Introduction 19
2.2 Problem Formulation 21
2.3 Definitions 22
3. Color-Based Detection Robust to Varying Illumination **25**
3.1 Introduction 25
3.2 Chromaticity Representation 26
3.3 Method 27
3.4 Experimental Results 31
3.5 Discussion 40
3.6 Conclusions 41
4. Compensation for Motion Blur **42**
4.1 Introduction 42
4.2 Motion-Blur Compensation 43
4.3 Experimental Results 48
4.4 Discussion 55
4.5 Conclusions 57
5. Initialization of the Kalman Filter **58**
5.1 Introduction 58
5.2 Preliminaries 60
5.3 Optimal Solution of a Linear System of Equations with Noise 61
5.4 Filter Initialization 62

5.5 Simulation Example 67
5.6 Discussion 71
5.7 Conclusions 73

6. Object Tracking **74**
6.1 Introduction 74
6.2 Problem Formulation 75
6.3 Method 76
6.4 Detection of False Positives 79
6.5 Results 81
6.6 Discussion 85
6.7 Conclusions 86

7. Robot Trajectory Generation **87**
7.1 Introduction 87
7.2 Problem Formulation 89
7.3 Method 89
7.4 Results 102
7.5 Discussion 109
7.6 Conclusions 111

8. System Integration of a Ball-Catching Robot **122**
8.1 System Overview 123
8.2 Image Analysis 124
8.3 Tracking 128
8.4 Robot Control 129
8.5 Accuracy Requirement 131
8.6 Performance 133
8.7 Discussion and Future Work 135
8.8 Conclusions 137

Part II Force Control and Estimation **139**

9. Robotic Assembly **141**
9.1 Introduction 141
9.2 Task Specification and Control Framework 143
9.3 Shield-Can Use Case 148
9.4 Emergency-Stop-Button Use Case 152
9.5 Experimental Results 165
9.6 Discussion and Lessons Learned 167
9.7 Conclusions 171

10. Robotic Force Estimation without any Force Sensor **172**
10.1 Introduction 172
10.2 Force Estimation Using Joint Control Errors 173
10.3 Force Estimation Using Motor Torques 181

10.4 Conclusions 197
11. Conclusions **201**
11.1 Vision-Based Control 201
11.2 Force Control and Estimation 202
11.3 Concluding Remarks 203
A. Fundamentals of Robotics and Computer Vision **204**
A.1 Homogeneous Coordinates 204
A.2 Robot Kinematics 208
Bibliography **210**

1

Introduction

1.1 Motivation and Background

Industrial robots are fast and accurate when working with known objects at precise locations in well structured manufacturing environments. In this classical automation setting, both peripherals/tooling and the programming are tailored to cope with commonly occurring variations. Under such conditions robots are outperforming humans in many applications, but at the expense of flexibility. Widening the perspectives, it is desirable that robots should be general-purpose working or human-assisting machines that perform well in real-world settings with time-varying, dynamic, and non-deterministic elements. To function in such environments the robots must be equipped with sensors, giving them the ability to measure the positions and other properties of the objects in their surroundings. They also need the cognitive ability to interpret the sensor information and react on it in a proper way.

Even in controlled factory environments there are benefits of adding sensing capabilities to robots. If the robots become good at handling uncertainties, less effort has to be put into structuring the environments around them. There are also limits on how accurately objects can be positioned or gripped. In assembly operations the relative positions of the objects to assemble are of interest, and they can be inferred with high accuracy from contact forces even in the presence of uncertainties in the absolute position measurements. The potential of using force sensing can be imagined by observing how humans outperform robots in many assembly operations, in spite of poor human position accuracy. The superior performance of humans can be explained by the dexterity of the hands, tactile sensing, impedance-control performance, cognitive vision, and the ability to detect and recover from error situations. Instead of the automation approach to robotics, developing productive robots that exhibit such sensor-based performance represents the robotics approach to automation.

This thesis presents work on how real-time sensor-based control can be used to gain improved performance and robustness of robotic task execution, considering both touch-based sensing (such as interaction forces between the robot and the work piece) and contact-less position sensing (such as vision). Particular attention is given to two challenging application cases: a ball-catching industrial robot, and force-controlled robotic assembly.

Several skills are required for catching a ball. The ball has to be detected and its position must be estimated and extrapolated to the future with high accuracy. Due to acceleration constraints, the robot has to start moving soon after an estimate of the catching position is available, and then the trajectory must be modified while moving, as the catching position is updated due to new measurements. All of these actions have to be performed under hard real-time constraints in order for the robot to reach the catching position before the ball reaches the robot.

Using force sensing in assembly provides important advantages over position-based control, but also introduces new challenges in terms of assembly strategies, task description and feedback-controller design. Due to the cost of force sensors, there are also incitements to be able to estimate the forces without using any force sensor.

Both of the above application examples require control based on high-rate sensor information, and together they represent some of the challenges in sensor-based robotic task execution.

1.2 Publications

Publications on Robotic Ball Catching

Chapters 2–8 are based on the publications below and some previously unpublished material. In these publications, M. Linderoth was the main contributor, and the co-authors assisted with discussion of the ideas and structuring of the manuscripts.

Linderoth, M., A. Robertsson, K. Åström, and R. Johansson (2009). "Vision based tracker for dart-catching robot". In: *Preprints 9th IFAC International Symposium on Robot Control (SYROCO'09)*. Gifu, Japan, pp. 883–888.

Linderoth, M., A. Robertsson, K. Åström, and R. Johansson (2010). "Object tracking with measurements from single or multiple cameras". In: *Proc. International Conference on Robotics and Automation (ICRA 2010)*. Anchorage, AK, USA, pp. 4525–4530.

Linderoth, M., K. Soltész, A. Robertsson, and R. Johansson (2011). "Initialization of the Kalman filter without assumptions on the initial state". In: *Proc. IEEE International Conference on Robotics and Automation (ICRA 2011)*. Shanghai, China, pp. 4992–4997.

Linderoth, M., A. Robertsson, and R. Johansson (2013). "Color-based detection robust to varying illumination spectrum". In: *IEEE Workshop on Robot Vision (WoRV 2013)*. Clearwater Beach, Florida, USA, pp. 120–125.

Publications on Robotic Assembly

Chapters 9–10 are based on the publications below, for which M. Linderoth and A. Stolt were the main contributors and assert equal contribution. The work was produced through close cooperation between the main authors, but M. Linderoth had a focus toward the theoretical aspects of the methods while A. Stolt had a focus toward implementation and experimental evaluation. A. Robertsson and R. Johansson assisted with discussion of the ideas and structuring of the manuscripts. The paper "Force controlled robotic assembly without a force sensor" received the Best Automation Paper Award at ICRA2012.

Linderoth, M., A. Stolt, A. Robertsson, and R. Johansson (2013). "Robotic force estimation using motor torques and modeling of low velocity friction disturbances". In: *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*. Tokyo, Japan.

Stolt, A., M. Linderoth, A. Robertsson, and R. Johansson (2011). "Force controlled assembly of emergency stop button". In: *Proc. IEEE International Conference on Robotics and Automation (ICRA 2011)*. Shanghai, China, pp. 3751–3756.

Stolt, A., M. Linderoth, A. Robertsson, and R. Johansson (2012). "Force controlled robotic assembly without a force sensor". In: *Proc. International Conference on Robotics and Automation (ICRA 2012)*. St. Paul, Minnesota, USA, pp. 1538–1543.

Stolt, A., M. Linderoth, A. Robertsson, and R. Johansson (2013). "Robotic assembly of emergency stop buttons". In: *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*. Tokyo, Japan.

Other Publications

Björkelund, A., L. Edström, M. Haage, J. Malec, K. Nilsson, P. Nugues, D. Störkle, A. Blomdell, R. Johansson, M. Linderoth, A. Nilsson, A. Robertsson, A. Stolt, and H. Bruyninckx (2011). "On the integration of skilled robot motions for productivity in manufacturing". In: *Proc.*

IEEE/CIRP International Symposium on Assembly and Manufacturing (ISAM2011). Tampere, Finland, pp. 1–9.

Stolt, A., M. Linderoth, A. Robertsson, M. Jonsson, and T. Murray (2011). "Force controlled assembly of flexible aircraft structure". In: *Proc. IEEE International Conference on Robotics and Automation (ICRA 2011)*. Shanghai, China, pp. 6027–6032.

Stolt, A., M. Linderoth, A. Robertsson, and R. Johansson (2012a). "Adaptation of force control parameters in robotic assembly". In: *10th International IFAC Symposium on Robot Control (SYROCO'12)*. Dubrovnik, Croatia, pp. 561–566.

Stolt, A., M. Linderoth, A. Robertsson, and R. Johansson (2012b). "Robotic assembly using a singularity-free orientation representation based on quaternions". In: *10th International IFAC Symposium on Robot Control (SYROCO'12)*. Dubrovnik, Croatia, pp. 549–554.

All of the above publications are available for download from `http://www.control.lth.se/Publications.html`.

1.3 Outline and Contributions

Part I, including Chapters 2–7, presents work done in the context of a robotic ball-catcher. Part II, including Chapters 9–10, describes work done in the context of force-controlled robotic assembly. Finally, conclusions are given in Chapter 11. Appendix A describes basic concepts from robotics and computer vision used in this thesis.

The chapters have been written so they should be possible to read independently. This comes at the cost of some repetition between the chapters.

Chapter 2 – Ball-Catching Robot

This chapter introduces the problem of robotic ball-catching, which serves as a common framework for the problems treated in Chapters 3–7.

Chapter 3 – Color-Based Detection Robust to Varying Illumination

When color is used for pixel classification in images, attention must be payed to varying illumination conditions. An object may be perceived to have different colors under different light sources. Chapter 3 presents a new systematic way of designing a color classifier that works for a desired range of illumination conditions.

Chapter 4 – Compensation for Motion Blur

When objects in a scene move during image exposure, there is a risk of motion blur appearing in the captured images. This presents a problem for color-based classification, since the colors in the blurred regions may resemble neither the foreground nor the background. Chapter 4 describes an original way of estimating which pixels were subject to motion blur on a pixel-by-pixel basis. This information is then used to improve how accurately the size and position of a moving object can be measured. The algorithm is very fast and suitable for real-time execution.

Chapter 5 – Initialization of the Kalman Filter

When using the traditional way of initializing the Kalman filter, a full estimate of the initial state has to be provided. The variance of the initial estimate may be large, but it has to be finite in all directions of the state space. Chapter 5 presents a novel way of initialization that makes no assumptions about the initial state and can produce a state estimate based purely on the measurements.

Chapter 6 – Object Tracking

Chapter 6 presents an alternative to traditional triangulation in stereo vision for dynamic objects. Instead, 2D image measurements are combined using a Kalman filter. When the dynamical model of the moving object is known, this approach makes it possible to estimate the state of the object even when no images are captured simultaneously, something that could not have been done using triangulation-based methods.

Chapter 7 – Robot Trajectory Generation

Chapter 7 presents a method for fast trajectory generation with the objective that the maximum acceleration should be minimized. The method is appropriate to use when the robot has to start moving before the destination position is completely known and the trajectory has to be modified while in motion.

Chapter 8 – System Integration of a Ball-Catching Robot

This chapter describes the integration of the components needed to form the ball-catcher system. The system was experimentally evaluated, and it demonstrated higher spatial accuracy and faster response time than ball catchers developed by other groups.

Chapter 9 – Robotic Assembly

Chapter 9 presents a framework for specifying and executing force-controlled robotic assembly. Experiences and lessons learned from applying the system to real-world assembly scenarios are presented.

Chapter 10 – Robotic Force Estimation without any Force Sensor

Force sensing is necessary for many robotic operations, but dedicated force sensors may be very expensive. Chapter 10 presents two new methods for estimating forces using the internal sensing of the robot. The first method presented uses the control errors of the robot joints to estimate the contact forces. The second method uses the motor torques and models that the uncertainty of the friction torques varies significantly, mostly due to varying joint speeds. This is used to improve the accuracy of the force estimates and provide dynamic confidence intervals in real time.

Part I

Vision-Based Control

2

Ball-Catching Robot

2.1 Introduction

Chapters 2–8 describe how high-speed computer vision can be used in a motion-control application. The problem considered is a ball-catching robot. An image of a robot catching a ball can be seen in Fig. 2.1. When a ball is thrown, the robot should move so that the ball hits the hole of a box mounted on the robot. The detection of the ball is to be performed with cameras providing data to the estimation of the position and future trajectory of the ball. In turn, this information should be used to move the box to the correct catching position.

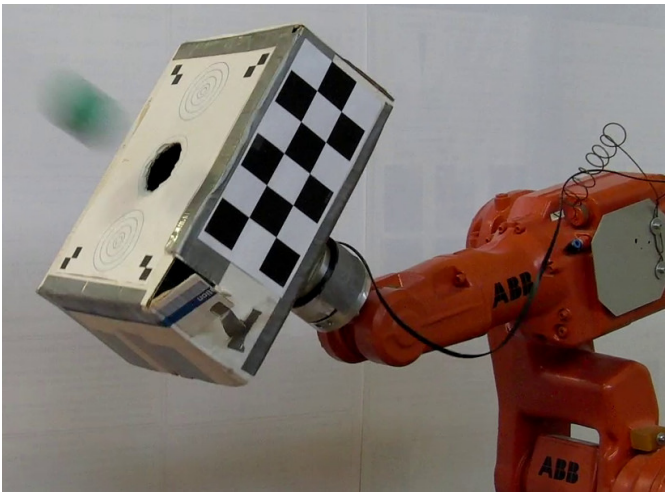


Figure 2.1 Image of a ball-catching robot with a green blurred ball flying toward the hole.

Robotic ball catching is a challenging task that requires several technologies working together, including fast image analysis, object tracking, and on-line trajectory generation. All these are capabilities that are required for robots that should perform reactive control in unstructured environments.

During the last two decades a number of ball-catching robot systems have been developed. The systems have used a range of different solutions when it comes to the positioning of cameras, whether the ball should be gripped by a robotic hand or caught in a different way, the type of robot etc. A few of them will be described below, and additional examples can be found in [Hove and Slotine, 1991; Hong, 1995; Riley and Atkeson, 2000; Barteit et al., 2008]

A ball-catching system at DLR [Frese et al., 2001] caught balls with 7 cm diameter in a net with 16 cm diameter. More recently, they used the mobile humanoid platform Rollin' Justin [Borst et al., 2009] to catch balls, and produced numerous publications on this topic, including [Birbach et al., 2008; Birbach and Frese, 2009; Bäuml et al., 2010; Birbach et al., 2011; Birbach and Frese, 2011; Bäuml et al., 2011b; Bäuml et al., 2011a], thus providing the best base for comparison with my system. They used a computationally relatively heavy motion planner, exploiting most joints of the robot, including moving the base. For vision they used high-resolution gray-scale stereo cameras mounted on the robot's head. The balls were detected using a generalization of the Hough transform [Ballard, 1981]. The camera placement on the head gave a relatively short baseline and caused the cameras to shake when the robot started to move to catch the ball. Hence, the motion of the head was estimated using an IMU during the catch.

The papers [Lippiello and Ruggiero, 2012a] and [Lippiello and Ruggiero, 2012b] described a ball catcher using a single eye-in-hand camera mounted in the palm of a robotic hand. When a ball was detected the robot made a sideways movement, thus getting views of the ball from different directions though the system only had a single camera. The trajectory of the ball was parameterized by its initial position and velocity, and the parameters were estimated by numerically solving an optimization problem that tried to minimize the errors between the simulated trajectory and the measurements. The control of the hand's position was decoupled from the control of its orientation to be able to always keep the ball in the field of view of the camera.

A hydraulic humanoid robot catching and tossing balls was described in [Kober et al., 2012]. It used an RGB-and-depth camera, observing the robot and the thrower from the side, to detect the ball with an accuracy of 5 cm. The robot was poorly damped, and they used a very simple linear kinematic model. The poor kinematic model alone resulted in po-

sition errors up to 5 cm within a catching range that was approximately 20 cm × 50 cm in a horizontal plane.

In [Bätz et al., 2010] basket balls were caught using a flat plate mounted on a robot. In the beginning of the catching motion, the plate was controlled to match the position and velocity of the ball. The ball was then smoothly decelerated to stay on the plate with out bouncing. Finally, the ball was balanced on the plate by means of a force sensor. The tracking of the balls was based on color and stereo vision.

This thesis presents work that aimed to develop a ball-catching robot with the highest possible position accuracy and fast response time. Much attention was given to the real-time performance of all components in order to cope with the timing requirements of the task. The work was to a large extent performed concurrently with the related systems described above.

2.2 Problem Formulation

The problem of robotic ball catching requires a range of sub-problems to be solved, some of which are listed below:

- image analysis
- outlier rejection
- tracking/state estimation
- ball-trajectory prediction
- choice of catching position
- robot trajectory generation

The list is not exhaustive and other ways of partitioning the problem are possible. For example, image analysis and state estimation can be integrated by exploiting the predicted position of the ball when performing the image analysis, or the initial pose and velocity of the robot can be taken into account when choosing the catching position. The listed problems can, of course, also be broken down into smaller problems. If color is used to detect the ball, the pixels should first be processed based on their color, followed by object detection.

Chapters 3–7 describe solutions to different components needed for the ball catcher. Chapter 8 then describes how the components were integrated to a complete system.

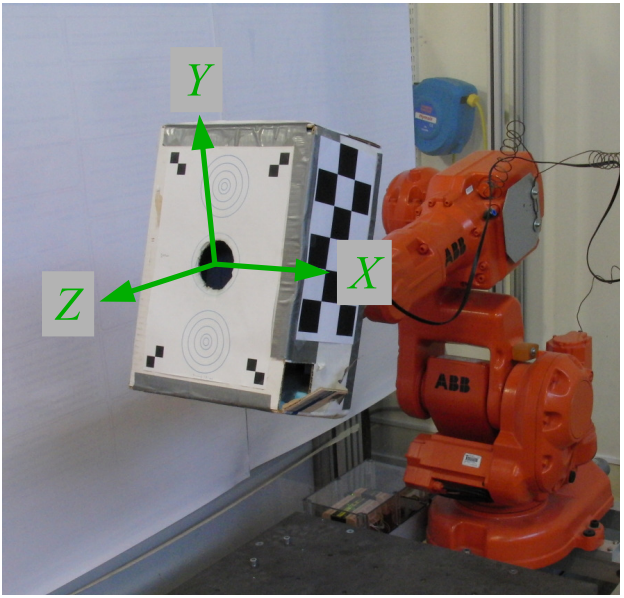


Figure 2.2 The coordinate system used to describe the position of the ball. The coordinate system is fixed w.r.t. the world frame and has its origin centered in the hole of the catching box when the robot is in its home position.

2.3 Definitions

This section describes a few concepts and definitions that will be used repeatedly in the subsequent chapters.

Coordinate system

The coordinate system used to describe the position of the thrown ball in this thesis is illustrated in Fig. 2.2. It is a right-handed Cartesian coordinate system with the y-axis pointing in the vertical direction and the z-axis in the horizontal plane pointing away from the robot. The origin of the coordinate system is centered in the hole of the catching box when the robot is in its home position.

Dynamic ball model

A ball in flight is considered to be affected by the gravity and by air drag proportional to the speed squared. The position of the ball, $p = [X\ Y\ Z]^T$,

in the coordinate system of Fig. 2.2 then follows the differential equation

$$\ddot{p} = -c\dot{p}\|\dot{p}\|_2 - \begin{bmatrix} 0 \\ g \\ 0 \end{bmatrix} + v_c, \tag{2.1}$$

where v_c is a load disturbance (e.g., acceleration caused by wind), g is the Earth gravity, and c is an air drag coefficient, depending on the size and mass of the ball, and the density of air, etc. With the state vector $x = [X \ Y \ Z \ \dot{X} \ \dot{Y} \ \dot{Z}]^T$ the differential equation (2.1) can be written in state-space form:

$$\dot{x} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -c\dot{X}V \\ -c\dot{Y}V - g \\ -c\dot{Z}V \end{bmatrix} + \begin{bmatrix} 0_{3\times 1} \\ v_c \end{bmatrix}, \tag{2.2}$$

where $V = \sqrt{\dot{X}^2 + \dot{Y}^2 + \dot{Z}^2}$ is the speed of the ball.

To discretize (2.1), let h be the sampling period, k the sample number, and t the time. Furthermore, we make the approximation that the acceleration is constant during each sampling interval, i.e.,

$$\ddot{p}(t) \approx \ddot{p}(kh) \quad \forall \quad kh \le t < kh + h. \tag{2.3}$$

This results in the non-linear discrete-time time-varying system

$$x(k+1) = \underbrace{\begin{bmatrix} 1 & 0 & 0 & h & 0 & 0 \\ 0 & 1 & 0 & 0 & h & 0 \\ 0 & 0 & 1 & 0 & 0 & h \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} \frac{h^2}{2} \begin{bmatrix} -c\dot{X}(k)V(k) \\ -c\dot{Y}(k)V(k) - g \\ -c\dot{Z}(k)V(k) \end{bmatrix} \\ h \begin{bmatrix} -c\dot{X}(k)V(k) \\ -c\dot{Y}(k)V(k) - g \\ -c\dot{Z}(k)V(k) \end{bmatrix} \end{bmatrix}}_{f(x(k))} + v(k), \tag{2.4}$$

where v is the discrete-time load disturbance. From now on the sampling period is left out from the time indices of discrete-time systems to make the notation shorter.

If the air drag is neglected ($c = 0$), then the system is linear and (2.4)

is an exact discretization of (2.2). The state update equation is then

$$x(k+1) = \underbrace{\begin{bmatrix} 1 & 0 & 0 & h & 0 & 0 \\ 0 & 1 & 0 & 0 & h & 0 \\ 0 & 0 & 1 & 0 & 0 & h \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\Phi} x(k) + \underbrace{\begin{bmatrix} 0 \\ -gh^2/2 \\ 0 \\ 0 \\ -gh \\ 0 \end{bmatrix}}_{\Gamma} + v(k). \qquad (2.5)$$

3

Color-Based Detection Robust to Varying Illumination Spectrum

3.1 Introduction

Whenever color recognition is used for object detection, varying illumination often causes problems, since an object may be perceived to have different colors under different lighting conditions. Many methods handle this difficulty by estimating the illumination spectrum and accounting for its effect on the perceived color. The method presented in this chapter uses a different approach and describes a systematic way to design a constant classifier that can detect a colored object for a given range of lighting conditions. This strategy also naturally handles the case where different parts of an object are illuminated by different light sources at the same time, e.g., sunlight from a window in combination with indoor lighting from the ceiling. Only one set of training data per light source has to be collected, and then the detector can handle any combination of the light sources for a large range of illumination intensities.

The fact that the apparent color of an object depends on the illumination is a well known problem that has been given a lot of attention in the field of color constancy [Ebner, 2007], which tries to estimate the 'true' color of an object irrespective of the illumination. Humans usually do this very well without realizing it, but there are also numerous examples [Lotto, 2011] of how the human vision system can be fooled. Land's retinex algorithm [Land, 1983] attempts to imitate the human vision system, and does so by finding large color gradients and looking at the relative color between nearby areas.

Another way to achieve color constancy is to look at the observed color of a reference object of known color and adjust the color of all observed

images, as done in [Balkenius et al., 2003]. If the illumination changes, a reference object has to be observed again to adjust the color correction. In [Soriano et al., 2000] the position and color of a face are tracked simultaneously to update the color range of the face as the illumination changes.

The methods described in the previous paragraphs depend on the environment (colors in the background or the availability of a reference color) and are adaptive in the sense that they can adjust their color correction if the illumination varies. The classifier presented in this chapter, however, is of the robust type in the sense that a single classifier works for a large range of illumination conditions. The method is illustrated by an example that can handle any combination of daylight and fluorescent light, which covers a large set of the conditions encountered in common indoor environments. The cost of having a single classifier for all lighting conditions is that the color range that is classified as foreground gets larger, which means that there is an increased risk of false positives, but if the range is chosen wisely the amount of false positives can be made small.

A number of different approaches to detecting balls for ball-catching robots have been presented by others. In [Bätz et al., 2010; Lippiello and Ruggiero, 2012a; Kober et al., 2012] color images were used and the balls were detected by defining ranges in the HSV color space [Forsyth and Ponce, 2002], but nothing was mentioned about handling of varying illumination spectrum. The positions of the balls were then estimated using the centroid of the detected blobs or the Hough transform [Ballard, 1981]. In [Barteit et al., 2008] gray-scale images were used and motion was detected by subtracting a background image from all images. The ball was then detected by means of the Hough transform. In [Birbach et al., 2011] the cameras were not stationary, so no background subtractions could be used. The balls were detected in high-resolution gray-scale images by using a generalization of the Hough transform.

3.2 Chromaticity Representation

A color can be described by its chromaticity and its luminance. The luminance represents the total intensity of the color. The chromaticity is independent of the luminance and only depends on the ratio of the red, green and blue intensities. In this chapter the chromaticity will be represented by the red and green channels in the normalized RGB space [Forsyth and Ponce, 2002]. Given a color (R, G, B), with $R, G, B \geq 0$, where the three components are the intensities of red, green and blue,

the chromaticity is given by

$$r = \frac{R}{R+G+B}, \qquad g = \frac{G}{R+G+B} \tag{3.1}$$

Now assume that we have two colors (R_1, G_1, B_1) and (R_2, G_2, B_2) with the corresponding chromaticities (r_1, g_1) and (r_2, g_2), and we make a linear combination of the two colors to form a third color:

$$(R_3, G_3, B_3) = \lambda_1(R_1, G_1, B_1) + \lambda_2(R_2, G_2, B_2) \tag{3.2}$$

with $\lambda_1, \lambda_2 \geq 0$. The resulting chromaticity is

$$\begin{aligned}
(r_3, g_3) &= \frac{(R_3, G_3)}{R_3 + G_3 + B_3} \\
&= \frac{\lambda_1(R_1, G_1)}{R_3 + G_3 + B_3} + \frac{\lambda_2(R_2, G_2)}{R_3 + G_3 + B_3} \\
&= \underbrace{\frac{\lambda_1(R_1 + G_1 + B_1)}{R_3 + G_3 + B_3}}_{\theta_1} \underbrace{\frac{(R_1, G_1)}{R_1 + G_1 + B_1}}_{(r_1, g_1)} \\
&+ \underbrace{\frac{\lambda_2(R_2 + G_2 + B_2)}{R_3 + G_3 + B_3}}_{\theta_2} \underbrace{\frac{(R_2, G_2)}{R_2 + G_2 + B_2}}_{(r_2, g_2)} \\
&= \theta_1 \cdot (r_1, g_1) + \theta_2 \cdot (r_2, g_2)
\end{aligned} \tag{3.3}$$

It can easily be verified that $\theta_1, \theta_2 \geq 0$ and $\theta_1 + \theta_2 = 1$, i.e., (r_3, g_3) is a convex combination of (r_1, g_1) and (r_2, g_2). This means that for all possible values of λ_1 and λ_2 in (3.2), (r_3, g_3) is on the straight line between (r_1, g_1) and (r_2, g_2). This property will be used in the design of the classifier proposed in this chapter.

3.3 Method

Outline of Method

The bullet lists below outline the suggested method, and the following subsections describe the steps in more detail.

Steps to be done off line

1. Collect training data with a single light source at a time. Collect many images from each scene and calculate the average to reduce the effect of noise.

2. Subtract bias from the images (the intensity recorded by the camera in complete darkness).

3. Make a histogram over the chromaticities. For each pixel in the training data, draw a line between the chromaticities at the different illuminations, and increase the value of all bins intersected by the line. The resulting histogram will be interpreted as a probability density function for the various different chromaticities.

4. Make a 3-dimensional table, where the inputs are the intensities of red, green and blue in a pixel, and the output is the probability density from the chromaticity histogram in the previous bullet. Add back the bias subtracted in step 2.

5. Blur the table from the previous bullet to account for image sensor noise.

6. Choose the probability density of the background.

7. For each color in the table, compare the probability densities of foreground and background to compute the probability of belonging to the foreground.

8. Calculate $(P(foreground) - 0.5)$ to get a *score* that is positive if the color is most likely to belong to the foreground and negative if the color is most likely to belong to the background. Put the scores in a look-up table to be used on line.

Steps to be done on line

1. Retrieve the score from the look-up table for each pixel in the image.

2. Find the circle that maximizes the sum of the scores of the pixels enclosed by the circle in the probability image (to detect balls if used for the ball-catching robot example).

Training-Data Collection

One set of images should be collected for each type of light source. For the example described in the introduction this means that two sets of images of the green balls should be collected; one where they are illuminated by daylight only, and one where they are illuminated by fluorescent light only. If possible, several images should be captured for each light source, to form the average image and reduce the effect of image sensor noise.

The light intensity returned by an image sensor is the sum of a bias and the actual light intensity. Depending on the type of sensor, the bias may be different for the different color channels and for different pixels.

Since a bias affects the ratios between the intensities of the different color components, the bias has to be subtracted before the chromaticities are calculated. Otherwise the chromaticity will depend on the luminance.

The averaging over many images and the bias correction are performed to be able to estimate the actual color distribution of the foreground objects. The variations that remain after these corrections stem mainly from differences in color on different parts of the training objects and from varying reflectance in different directions.

Histogram

A histogram over the chromaticities in the training data is created to estimate the probability density (up to a scale) for different chromaticities. For this histogram a grid of bins is created in the chromaticity plane.

For each pixel in the training data, let the bias-corrected intensities be (R_1, G_1, B_1) and (R_2, G_2, B_2) for the two light sources respectively, and the corresponding chromaticities be (r_1, g_1) and (r_2, g_2). According to the last paragraph in Sec. 3.2, the observed chromaticity will then be on the straight line between (r_1, g_1) and (r_2, g_2) for all combinations of the two light sources. Hence, all bins in the histogram that are intersected by the line between (r_1, g_1) and (r_2, g_2) are increased by 1. When all pixels have been added to the histogram, it is normalized so the total probability mass equals 1.

Look-Up Table

In order to minimize the amount of calculations at classification time, as much as possible is calculated off-line and stored in a look-up table. The table takes the intensities of red, green and blue in an image as inputs and returns the probability of that pixel belonging to the foreground.

When choosing the number of elements in the look-up table there is no reason to choose more elements than the color depth of the image can produce. Reducing the number of intensity levels, however, may give significant reduction of the size of the look-up table (scaling as $O(n^3)$), with only marginal reduction in the color classification accuracy caused by larger round-off errors in the intensities.

As a first step in creating the look-up table, the bias is subtracted from the (R, G, B)-intensities of each element and the corresponding probability density is retrieved from the (r, g)-histogram. Each element in the look-up table covers a small cube in (R, G, B)-space, and to reduce sampling artifacts the resulting probability density is calculated as the average of all bins in the (r, g)-histogram covered by the small (R, G, B)-cube. In this process the probabilities are assumed to be independent of the total intensities, $(R + G + B)$. The result will be a look-up table where

most elements are close to zero, and the elements with a high probability of belonging to the foreground will form a cone with its tip at the point corresponding to the bias, and the cone gets a larger cross-section area as the total intensity increases.

We now have a table with the probability densities for the actual colors of the foreground objects, but the captured images will be corrupted by noise. Hence, the look-up table is convoluted with a Gaussian kernel that corresponds to the noise variance of the image sensor. This smoothing operation handles in a neat way the fact that the chromaticity is affected more by noise in dark areas than in well illuminated areas. In the parts of the look-up table where the intensities are high, the cone of foreground colors is wide compared to the smoothing kernel and the probability densities will not be affected much by the smoothing operation. However, in the darker parts of the table, near the tip of the cone, the smoothing operation will spread out the probability mass over a much larger volume than before and will result in a lower peak probability density. This reflects the reality well and results in more uncertain classifications in dark areas.

We now have a table with scaled probability densities for observing different colors from a foreground object. To determine the probability P_{fg} of the pixel belonging to the foreground, the probability density p_{fg} of the foreground has to be compared to the probability density p_{bg} of the background:

$$P_{fg} = \frac{p_{fg}}{p_{fg} + p_{bg}} \tag{3.4}$$

For simplicity, the probability density of the background is assumed to be the same for all colors, and its value is the only tuning parameter of the algorithm.

For the elements in the look-up table where any of the colors is saturated, the probability is set to $P_{fg} = 0.5$, indicating that nothing can be said about that pixel and both foreground and background are equally likely.

Finally, all elements in the look-up table are subtracted by 0.5 to generate scores. This results in a table where positive scores indicate that the color is most likely to belong to the foreground and negative scores indicate that the color is most likely to belong to the background.

Image of Scores

The operations described so far can be done off line during the training phase. The remaining operations have to be done in real time on the images that should be analyzed. The first operation on the image is to replace the RGB-value of every pixel with the corresponding score from

the look-up table. This gives a soft foreground/background classification of each pixel.

Motion Detection

Motion detection may be performed if it is desired to only detect objects that are moving. For the proposed motion-detection method it is assumed that the image sensor noise is Gaussian with known variance. For each pixel, the difference between the current image and the previous image are calculated. If the difference is small, it was probably only caused by noise, but if the difference is big, there was probably motion in the area exposing that pixel. Let P_m be the confidence level (in the range $[0, 1]$) of a hypothesis test testing whether the difference could be inferred to be caused by motion, and not only by noise.

If the score of the pixel is positive, it is multiplied by P_m. If P_m is close to one (there was almost certainly motion), the score is hardly affected. If P_m is close to zero, motion can neither be confirmed nor rejected, and the score is reduce toward the neutral value zero.

Since only positive scores are affected by the motion detection, and since they are reduced toward 0 (not -0.5) objects can be detected even if parts of the background behind the foreground object have a color similar to the foreground object.

Object Localization

For the application of the ball catcher it is necessary to localize thrown balls in the images. For this purpose, each image is searched for circles that locally maximize the sum of the scores for all pixels enclosed by the circle. If the circle is made larger than the optimum, negative scores will be included and reduce the sum. If the circle is made smaller than the optimum, fewer pixels with positive score will be included, and the sum will be reduced. The optima can be found efficiently using integral images [Viola and Jones, 2001].

3.4 Experimental Results

Cameras

The algorithm described in this chapter was experimentally verified on images captured by a Basler A602fc camera, connected to the computer via FireWire. All images used for the experiments were captured with the intensities as integers in the range $[0, 255]$.

Training

The bias of the camera was measured by capturing a number of images in complete darkness and calculating the average intensities. The bias was $(5.5, 5.7, 41.4)$ and the standard deviation of the sensor noise was $(2.1, 1.5, 3.8)$, for the red, green and blue channels respectively. No significant differences could be observed between the different pixels.

Seven green balls were illuminated by fluorescent light only, and 100 images were captured by the camera. By calculating the average image the colors could then be calculated with a standard deviation of $(0.21, 0.15, 0.38)$. Similarly 100 images were captured at noon when the balls were illuminated by daylight only. One example image for each light source is shown in Fig. 3.1. Seven green balls, marked by white circles in the figure, were manually picked out and the 1859 pixels enclosed by those circles were used for training. Note that the remaining green balls in the image had a slightly more yellowish green color.

Figure 3.2 shows a scatter plot of the chromaticities for all the pixels in the training data. Note that the sets of points from the different light sources were almost disjoint. This means that if a classifier was trained for only one of the light sources, it would work very poorly for the other light source. For the histogram 750×750 bins were used and the result is shown in Figs. 3.3 and 3.4. In Fig. 3.4 you can see the lines drawn between the chromaticity pairs from the different illuminations, most in the direction from lower left to upper right. The probability density of the background was chosen so the total probability of the background was 6 times larger than that of the foreground. This value was selected

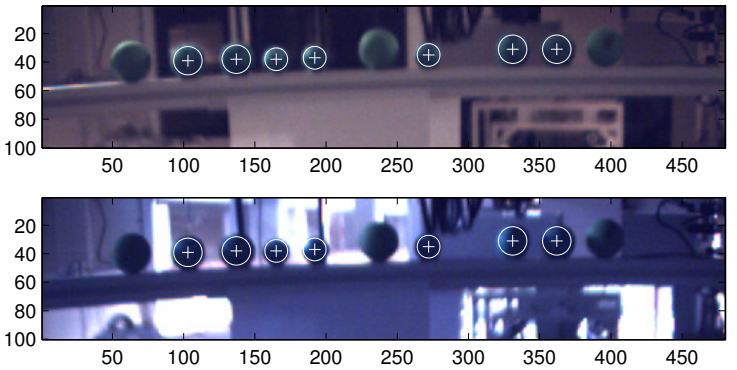


Figure 3.1 Sample images from the training data. The top image was captured with fluorescent light and the bottom image was captured with daylight. The pixels enclosed by the white circles were used for training.

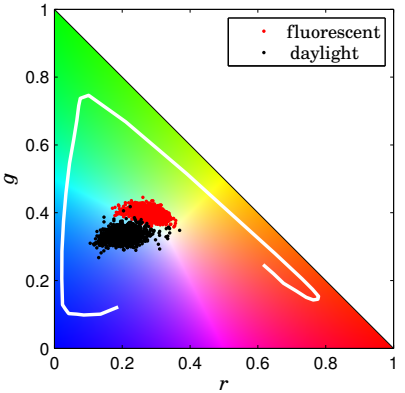


Figure 3.2 Scatter plot of the chromaticity for the pixels in the training data. The black and red points correspond to data from the two different light sources. The color scale in the background illustrates the chromaticities of the points. For the camera used, the white curve illustrates the response to monochromatic light in the range 390 – 700 nm.

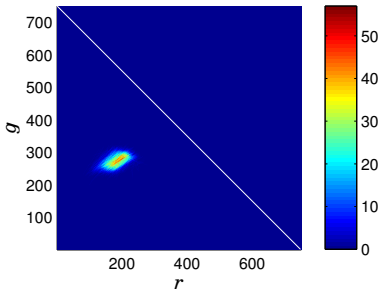


Figure 3.3 Histogram of foreground colors in the chromaticity space.

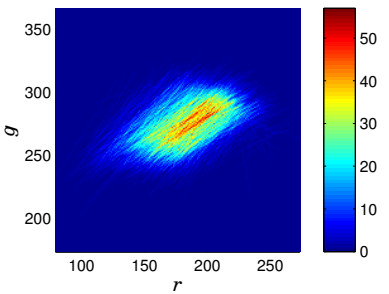


Figure 3.4 The same histogram as in Fig. 3.3, zoomed in on the area with the observed chromaticities.

by tuning to find a good trade off between missed detections and false positives.

Different sizes of the look-up table were evaluated and it was concluded that the number of intensity levels per channel could be reduced from 256 (as in the original images) to 64 without any significant decrease in the performance, giving a table size of $64^3 = 262144$ elements. Figures 3.5 and 3.6 show scatter plots of the elements that had probability densities higher than the background before and after smoothing. The probability densities were transformed to probabilities and subtracted by 0.5 to get

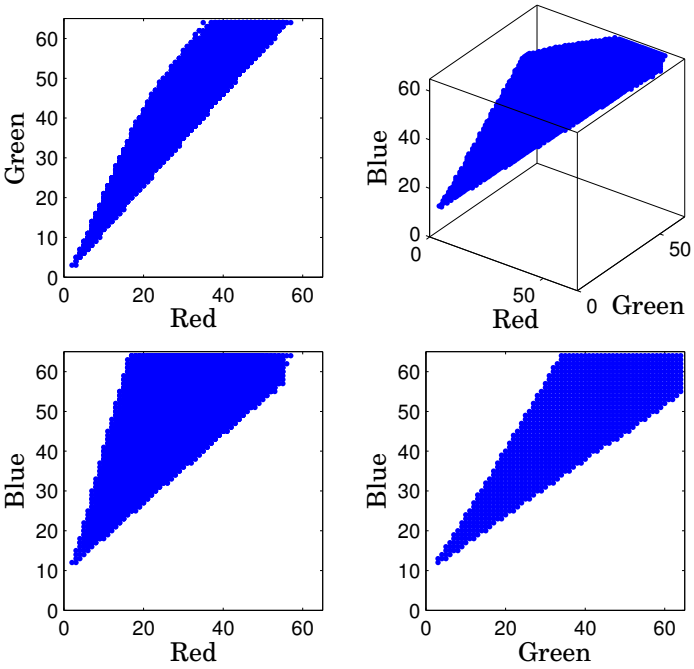


Figure 3.5 Scatter plot of the elements in the look-up table that had higher probability density than the background before the smoothing, viewed from four different directions. Note that the values were continuous in the range $[0, 1]$, though this plot only shows which elements exceeded the threshold 0.5. The tip of the cone corresponds to the bias of the camera.

the scores. The scores were scaled from the range $[-0.5, 0.5]$ to $[-127, 127]$ and stored in 8-bit signed integers in the look-up table. The resulting size of the look-up table was hence 256 kB, which easily fits inside the L2 cache of modern processors, allowing fast access when the pixels of an image were classified.

Classification

Images were collected under various different illumination conditions to test the performance of the classifier. In Fig. 3.7 one can see three images captured with fluorescent light, daylight or a combination of both. In the upper part of each image one can see a number of green balls placed on a bar. The seven middle balls (of the type that was used for training) had a slightly more bluish color than the two outer balls, that were more yellowish green. In every image you can also see a ball (of the bluish kind

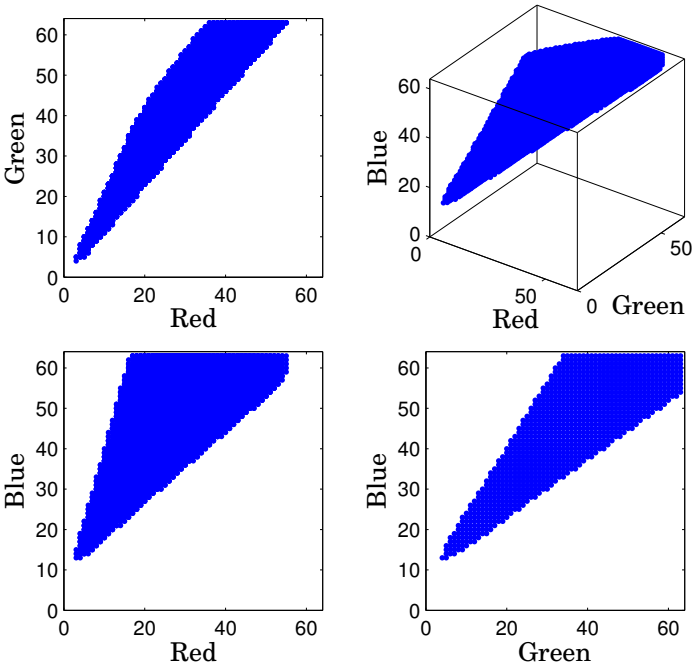


Figure 3.6 Scatter plot of the elements in the look-up table that had higher probability density than the background, after the smoothing. There is a small but important difference in smoothness compared to Fig. 3.5

that was used for training) thrown toward the area below the camera.

If you look at different areas in the images, you can see that dark areas tended to have probabilities close to 0.5, since the sensor noise made it hard to determine the chromaticity of the object, while bright areas tended to have probabilities close to one if they were green balls and close to zero otherwise. The pixels where any of the color channels was saturated had the probability 0.5.

The classification was the most confident when the scene was illuminated by a combination of daylight and fluorescent light. This can be understood by looking at Figs. 3.2 and 3.4. The center area in Fig. 3.4 had the highest probability density, since those colors were likely to be observed for all illumination conditions. When only one light source was used, more of the observed colors were close to the boundary with lower probability density.

Figure 3.8 illustrates how the range of colors classified as foreground

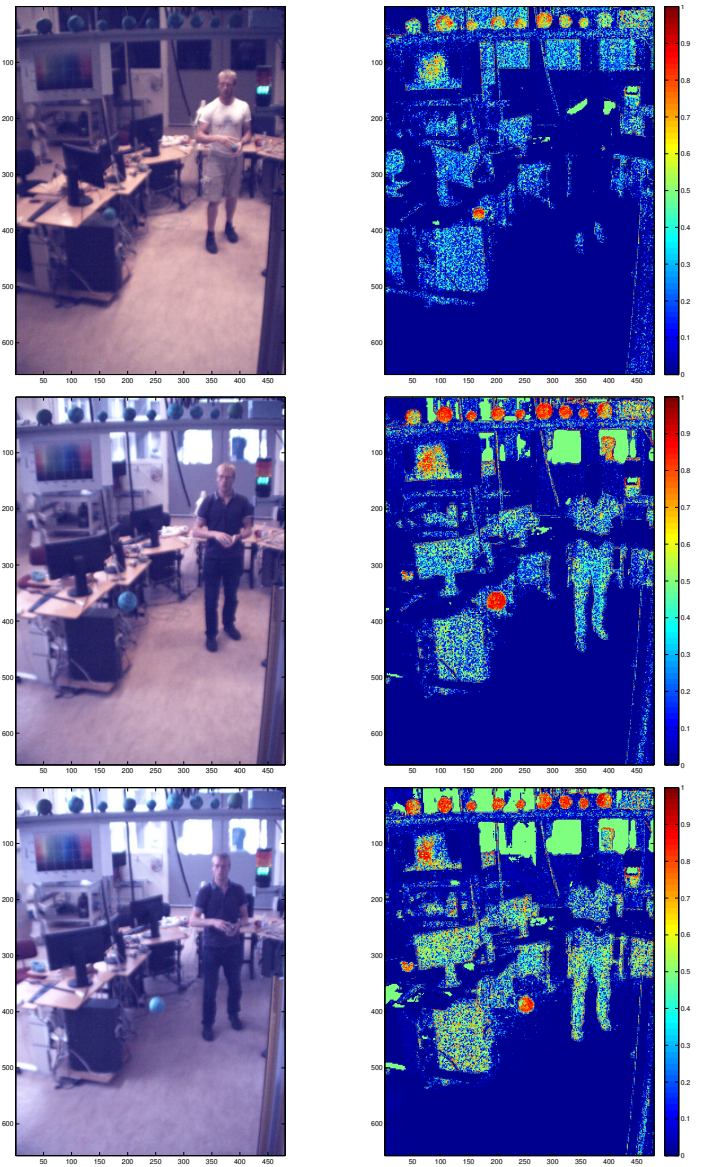


Figure 3.7 *Left column:* Images that were used to test the classification performance. *Right column:* Images showing the probability of a pixel belonging to the foreground. *Top row:* Fluorescent light only. *Middle row:* Fluorescent light and daylight. *Bottom row:* Daylight only.

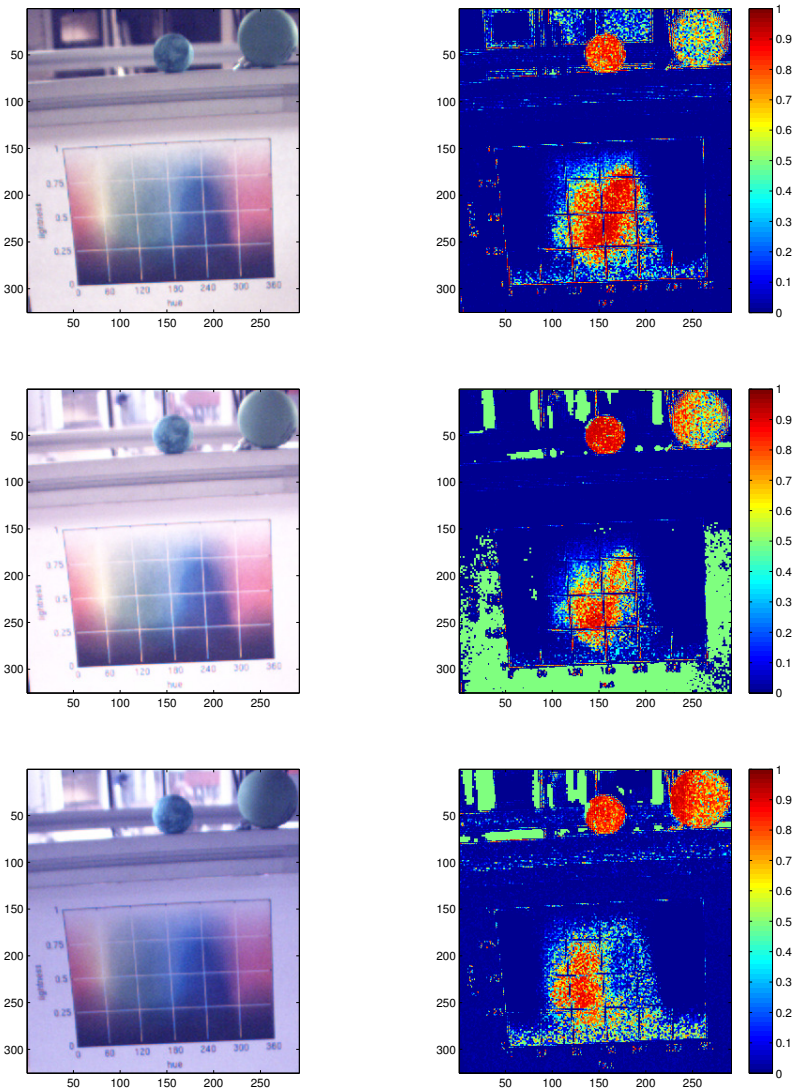


Figure 3.8 Images of color scales, illustrating the width of the classifier in different illumination conditions. *Top row:* Fluorescent light only. *Middle row:* Fluorescent light and daylight. *Bottom row:* Daylight only.

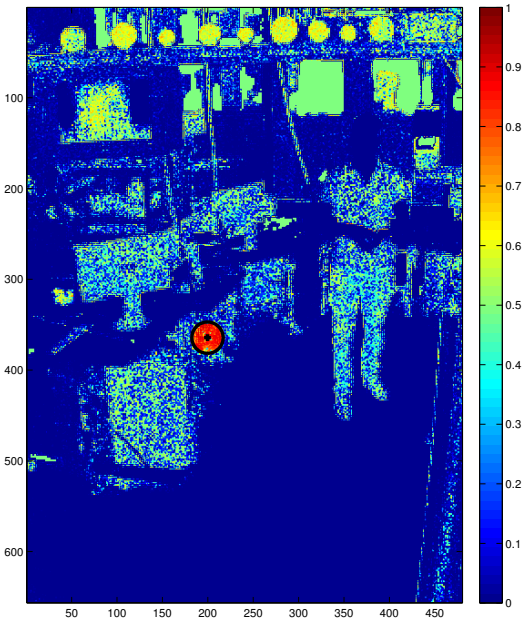


Figure 3.9 Probability of belonging to the foreground, where the probability was reduced for the pixels that did not change much since the previous image.

varied with the lighting conditions. In the yellowish fluorescent light in the top row, the cyan color in the color scale was classified as foreground, since the observed color was close to that of the bluish green ball in daylight. In the bluish daylight, however, the cyan color in the color scale was too blue to be classified as foreground. On the other hand, the bluish daylight made the yellowish large ball appear so blue that it was classified as foreground, as seen in the bottom row of Fig. 3.8. The small green ball (of the bluish kind used for training), was confidently classified as foreground under all illuminations.

The images shown in Fig. 3.7 are samples from image sequences captured with a frame rate of 50 fps. Figure 3.9 shows the resulting probability image when the middle image in Fig. 3.7 was compared to the previous image to perform motion detection, and the probabilities were reduced for the pixels that had not changed much since the previous image. Then the thrown ball was the only object with a high probability of belonging to the foreground. The result of the ball detection described in the last paragraph of Sec. 3.3 is indicated by a circle.

Ball-Catching Robot

The algorithm described in this chapter was successfully used to let a robot catch tossed balls, as described further in Chapter 8. The execution time for the entire image analysis process was approximately 5 ms per 656×480-pixel image when executed on one of the cores of an Intel® Core™2 Quad CPU Q6600 processor running at 2.40 GHz

Validation of Assumptions

The method described in this chapter assumes that all illuminations are linear combinations of two colors. To investigate whether this well describes the illumination conditions commonly encountered in real life, images of the green balls were captured under a number of different illumination conditions. The resulting distributions of the chromaticities are shown in Fig. 3.10. The images were captured in a lab with windows, but no direct sunlight could hit the balls. The daylight data used to train the classifier for the experiments described previously in this section (green in Fig. 3.10) were captured when the sky was blue but the sun itself was behind clouds. The fluorescent-light training data (blue in Fig. 3.10) were captured at night with fluorescent ceiling lamps pointing upwards, so most of the light hitting the balls was reflected off the white ceiling.

The additional data sets were captured when it was sunny (red), cloudy

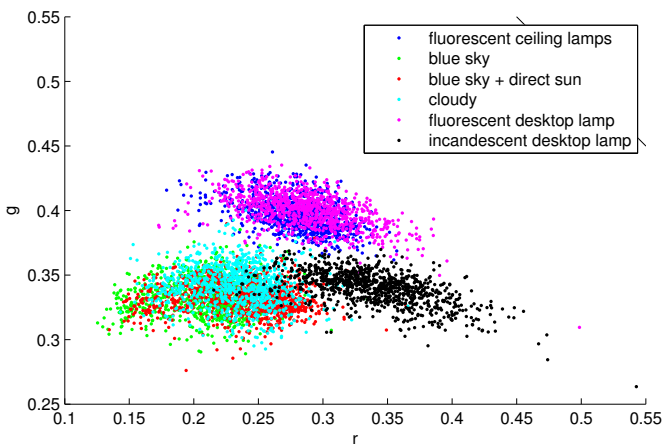


Figure 3.10 Observed chromaticities of the green balls for different illumination conditions.

(cyan), with a fluorescent desktop lamp shining directly onto the balls (magenta), and with a light bulb shining directly onto the balls (black). Note that in Fig. 3.10 the different data sets were plotted one after the other, so the data sets in the bottom of the legend partly hide the data sets in the top of the legend.

It can be seen that the different kinds of daylight and fluorescent light almost formed a line in the chromaticity space. The fluorescent desktop light was almost identical to the ceiling light that was used for training. The cloudy light was close to the sunny light, but slightly further toward the chromaticities of the fluorescent light. The light bulb, however, gave chromaticities that were quite different.

Figure 3.10 indicates that the illuminations encountered in practice really can be described as combinations of sunlight and fluorescent light, as long as light bulbs are not used.

3.5 Discussion

In this chapter the probability density was assumed to be uniform for all colors in the background, which is quite an unrealistic assumption. As an alternative the background distribution could have been estimated in the same way as the foreground, but then there would be a big risk that the classification would perform poorly if the camera was moved to a different room with different colors in the background. The uniform background distribution was used here to show that the method worked with such simple assumptions and should work even better with a properly modeled background.

Only a combination of *two* light sources was considered. The method could easily be generalized to several light sources. Instead of drawing a line between the data points from the two light sources, one should then draw the polygon forming the convex hull of the data points.

It was assumed that nothing was known about a pixel if any color channel was saturated, which was conservative. For example, if only the red and blue channels were saturated, one could with high confidence say that the object was not green.

Many different machine learning strategies could have been considered for training the classifier [Bishop, 2006], but if they are applied without a good understanding of the problem it may be hard to collect data that cover all illumination conditions that can be encountered at classification time. With the method described in this chapter it was enough to capture data under two different illumination conditions to be able to classify images from a large range of illumination spectrums and a range of illumination intensities that was limited only by the dynamic range of the

camera.

3.6 Conclusions

A method for designing a color classifier that works over a large range of illumination conditions was presented. Only two different illumination conditions were required during the training phase and there is only one tuning parameter. The method was experimentally verified on real data and used to detect balls for a ball-catching robot.

4

Compensation for Motion Blur by Extrapolation of Color Change

4.1 Introduction

When color is used to detect a moving foreground object in an image with motion blur, the blurred regions may be difficult to classify correctly, since the resulting colors may be significantly different from both the foreground and the background. As a consequence, the position and size of the foreground object may be difficult to estimate accurately.

This chapter presents a method for classifying the pixels in the blurred regions more correctly. The color of the current image is compared to a background image to estimate which pixels were subject to motion blur and how large fraction of the exposure time they were exposed by the foreground object. This makes it possible to determine the pose of the foreground object more accurately. The method executes very fast and is appropriate to use for real-time tracking.

The problems of image matting and blending are essential for cutting an object from one image and pasting it into another [Jia et al., 2006; Lalonde et al., 2007; Levin et al., 2008]. The color distributions of both foreground and background, the object contour, and the level of transparency of the edge pixels are then estimated, often using very little prior information. At the edges of the object (in particular for fuzzy edges, e.g., around hair) the colors of foreground and background mix, which in many ways resembles the process of motion blur. The methods that can solve these general problems are, however, not suitable for real-time image processing at high frame rates.

Much attention has been given to the problem of restoring blurred images [Harris, 1966; Lu et al., 2006; Sezan and Tekalp, 1990; Yitzhaky

et al., 1999]. These methods typically model the relative motion between the camera and the scene by estimating a point spread function (PSF). The observed image is assumed to be generated by convolution between an ideal image and the PSF. An estimate of the ideal image is then recovered by deconvolution of the blurred image, taking noise characteristics into account.

When objects should be detected, however, it is not necessary to reconstruct the original image. This fact is exploited by the method described in this chapter, which instead analyzes the images to determine which pixels were subject to motion blur and estimates what fraction of the exposure time they were exposed by a moving object. This information can be used to determine the pose of the moving object more accurately. A key property is that the method should be very fast, so that it can be used for object tracking in real time.

Object tracking in videos has been treated in a large number of papers, e.g. [Perez et al., 2002; Shen et al., 2010; Wu et al., 2009]. In [Dai et al., 2006; Wu et al., 2011] the tracking performance was improved by explicitly modeling the motion blur. These methods are more general than the method presented here, in terms of what foreground objects they can track and what backgrounds they can handle. They are, however, also more complex and require much longer computation times.

The method described in this chapter was experimentally evaluated by detecting balls for the ball-catching robot described in Chapters 2 and 8. The same problem has been treated in [Bätz et al., 2010; Lippiello and Ruggiero, 2012a; Kober et al., 2012], which used color-based detection, and in [Barteit et al., 2008; Birbach et al., 2011], which used gray-scale images and variations of the Hough transform. None of them, however, mentioned any specific handling of motion blur.

4.2 Motion-Blur Compensation

Problem Formulation

In this chapter, I address the problem of determining whether the color of a pixel may be the result of motion blur, under the requirement that it should execute in real time for high frame rates. Based on such analysis, a method is designed to re-evaluate which pixels could be part of a foreground object. Application of this method permits the pose of moving foreground objects to be estimated more accurately.

Assumptions

The work in this chapter assumes that static cameras are used to capture images of moving foreground objects. There should be a way to estimate a

background image. The background should be fairly static but can contain moving background objects.

There should be a function, $0 \leq P_{fg}(c) \leq 1, \forall c$, that maps the color, c, of a pixel to the probability that the pixel belongs to the foreground (when there is no motion blur). The term *foreground colors* is in this chapter used to denote colors for which P_{fg} is large, though P_{fg} is continuous and there is no sharp boundary for the foreground colors. The set of foreground colors should be convex in the space of color component intensities, explained further on page 47.

Fundamentals of Motion Blur

During image exposure each pixel is assumed to be exposed to its background color during the time $T_{bg} \geq 0$ and exposed to a moving object during the rest of the exposure; $T_{mv} \geq 0$. They relate to the total exposure time, T, as

$$T = T_{bg} + T_{mv}. \tag{4.1}$$

Either T_{bg} or T_{mv} could be zero, which corresponds to the case with no motion blur.

It is important to note that nothing has been assumed about the color of the moving object at this point. There can be many moving objects in the scene, but only (moving or static) objects within a given color range are considered to be foreground objects in the terminology of this chapter.

Let c_{bg} be the color of a given pixel in the background image, represented as a vector with the intensities of the different color components, e.g., red, green, and blue. Similarly, let c_{mv} be the color of a moving object in the scene. An example with two color components is shown in Fig. 4.1.

If the moving object is passing in front of a given pixel during the exposure, that pixel may be partly exposed by the background and partly exposed by the moving object. The resulting observed color, c_{obs}, will then be a convex combination of c_{bg} and c_{mv}:

$$c_{obs} = \frac{T_{bg}}{T_{bg} + T_{mv}} c_{bg} + \frac{T_{mv}}{T_{bg} + T_{mv}} c_{mv} \ . \tag{4.2}$$

Fig. 4.1 illustrates how the color c_{obs} is created. The relation between the exposure times and the distances, a and b, in the figure is given by

$$\frac{a}{T_{mv}} = \frac{b}{T_{bg}} \ . \tag{4.3}$$

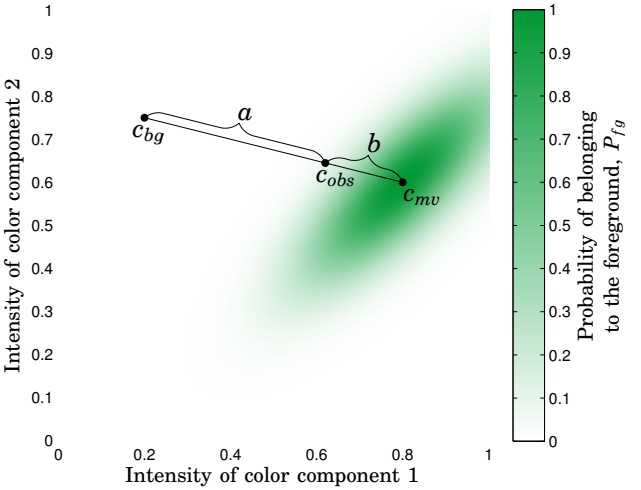


Figure 4.1 Illustration of how the observed color is generated in a pixel when motion blur occurs. In this example, two color components are used. c_{bg} is the background color for the given pixel, and c_{mv} is the color of a moving object that projects onto the pixel during part of the exposure. c_{obs} is the observed color resulting from the motion blur. The green/white shades show how likely it is that a pixel belongs to the foreground, as a function of the color.

Let r be the fraction of the exposure time that the pixel was exposed by c_{mv}, calculated as

$$r = \frac{T_{mv}}{T_{mv} + T_{bg}} = \frac{a}{a + b} \ . \tag{4.4}$$

Figure 4.1 demonstrates an example scenario. The green/white shades show P_{fg}, the probability of a pixel belonging to the foreground as a function of the color. Further, c_{mv} is in the center of the region of foreground colors, and the exposure times relate as $T_{mv} : T_{bg} = a : b = 70 : 30$. This example, hence, simulates a moving foreground object. Even though the pixel was exposed by a foreground color during $r = 70\%$ of the exposure time, the observed color is classified as being almost certainly background, since c_{obs} is outside the green area in the figure.

Method

When the motions of objects in the scene are not known, only c_{bg} from the background image and c_{obs} from the current image are available, and

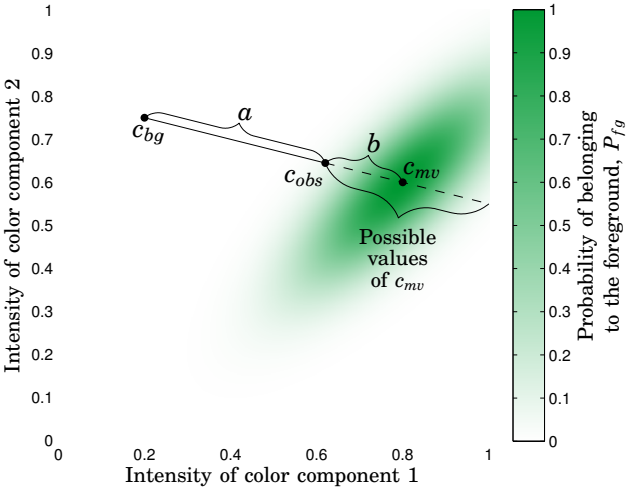


Figure 4.2 When the method is used, the background color, c_{bg}, and the observed color, c_{obs}, are known. c_{obs} is possibly generated by mixing c_{bg} with some color, c_{mv}. If the ray of possible values of c_{mv} intersects the region of colors that have a high probability of being foreground, it is possible that the pixel was exposed by a foreground object during a part of the exposure time.

the values of c_{mv}, r, and b etc. have to be estimated. Using the model (4.2) for how colors mix, c_{obs} must be on the line between c_{bg} and c_{mv}, as illustrated in Fig. 4.1. Equivalently, when c_{bg} and c_{obs} are the colors known, c_{mv} must be somewhere on the ray that extends from c_{obs} in the direction opposite to c_{bg}, as illustrated by the dashed line in Fig. 4.2. The ray of possible values of c_{mv} can be interpreted as an extrapolation of how the color changed from the background, c_{bg}, to the current observation, c_{obs}.

Hence, the set of possible values of c_{mv} that could have generated the observation, c_{obs}, is restricted to a line, and each hypothesis of c_{mv} is associated with a corresponding value of r, which can be used to parameterize the hypotheses. Inserting (4.4) into (4.2), and solving for c_{mv} gives

$$c_{mv}(r) = \left(1 - \frac{1}{r}\right) c_{bg} + \frac{1}{r} c_{obs}. \tag{4.5}$$

An alternative way to determine c_{mv} as a function of r is to solve (4.4)

for b,

$$b(r) = \left(\frac{1}{r} - 1\right) a, \tag{4.6}$$

and then use the distance b to determine c_{mv} graphically from a figure like Fig. 4.2. The case $b(1) = 0$ corresponds to the pixel being exposed only by a moving object and $c_{mv} = c_{obs}$. Values of r closer to zero correspond to larger T_{bg} and larger values of b, i.e., c_{mv} being extrapolated further away from c_{obs} and c_{bg}.

If the ray of possible values of c_{mv} intersects the region of colors where P_{fg} is large, it is possible that the observed color was the result of mixing the background with a foreground object. The task is now to estimate c_{mv} and an associated probability of belonging to the foreground.

It is unlikely that both r and $P_{fg}(c_{mv}(r))$ are large simultaneously unless c_{obs} was generated by mixing c_{bg} with a foreground color, as we will see in the next subsection. Motivated by this insight we propose that the estimate of r should be given by

$$\hat{r} = \underset{0<r\leq 1}{\arg\max} \quad r \cdot P_{fg}(c_{mv}(r)). \tag{4.7}$$

In the presence of motion blur, P_{fg} is not appropriate to answer the question whether a pixel is part of the foreground, since it has no clear interpretation when the pixel was exposed by foreground only during a fraction of the exposure. To find the pose of the foreground object, we propose to instead use the quantity

$$\begin{aligned} R_{fg} &= \hat{r} \cdot P_{fg}(c_{mv}(\hat{r})) \\ &= \max_{0<r\leq 1} r \cdot P_{fg}(c_{mv}(r)), \end{aligned} \tag{4.8}$$

where the probability that the moving object is a foreground object is weighted by how long the pixel was exposed by it. It has the property that the size and shape of the region where $R_{fg} > 0.5$ is similar to the actual foreground object.

Influence of Disturbances

Foreground objects with non-uniform color The measurement model (4.2) is a simplification of the real measurement process. Specifically, it assumes that the moving object has uniform color.

If the moving object has non-uniform color, it will generate the same observation, c_{obs}, as if the entire moving object had the color $\bar{c}$, where $\bar{c}$ is the average of (and, hence, a convex combination of) the colors that projected to the pixel during the exposure.

This means that if a pixel is exposed by a range of foreground colors, the measurement model (4.2) approximates these colors by a convex combination thereof. If P_{fg} has convex superlevel sets, this convex combination also is a foreground color. The method, hence, works for objects with non-uniform colors, but the set of foreground colors (for example the green area in Fig. 4.2) should be convex. However, having a too large range of foreground colors also increases the risk of false positives.

Motion in the background There is a risk that moving objects in the image are erroneously classified as foreground. This can happen if the color of the moving object is positioned between the colors of the background and the foreground in the color space. For example, the measurement c_{obs} in Fig. 4.2 could be generated by the marked hypothesis of c_{mv} and $r = 0.7$, but it could also be generated by a moving object with $c_{mv} = c_{obs}$ and $r = 1$. In order for this situation to generate large values of R_{fg}, however, both of the following conditions must be fulfilled:

1. The color of the moving object is positioned between c_{bg} and the foreground colors.
2. The color of the foreground object is close to the foreground colors.

If the first condition is not fulfilled, $P_{fg}(c_{mv}(r))$ is small for all r. If the second condition is not fulfilled, r is small. In either case, the maximand of (4.8) is small, thus, not generating any false positive of R_{fg}.

Image sensor noise The measurement model (4.2) implies that if $c_{obs} \neq c_{bg}$, an object has moved in front of the pixel. In practice, the difference in color can also be caused by image sensor noise, i.e., $c_{obs} = c_{bg} + e$, where e is noise.

For the noise to cause false large values of R_{fg}, the randomly generated ray of possible c_{mv} has to point in the direction of the foreground colors. How likely this is depends on the size of the region of foreground colors, and it is less likely to happen the more color components are used.

When $c_{obs} = c_{bg} + e$ and c_{bg} is not similar to the foreground colors it is required that $b \gg a$ (i.e., r small) to find a hypothesis of c_{mv} with a large value of $P_{fg}(c_{mv})$. When extrapolating the color so far, however, the small value of r makes the maximand of (4.8) small.

The sensor noise is, hence, unlikely to generate false positives unless the background color is very similar to the foreground colors.

4.3 Experimental Results

The method described in this chapter was evaluated on data from the ball-catching robot system described in Chapters 3 and 8. The cameras used

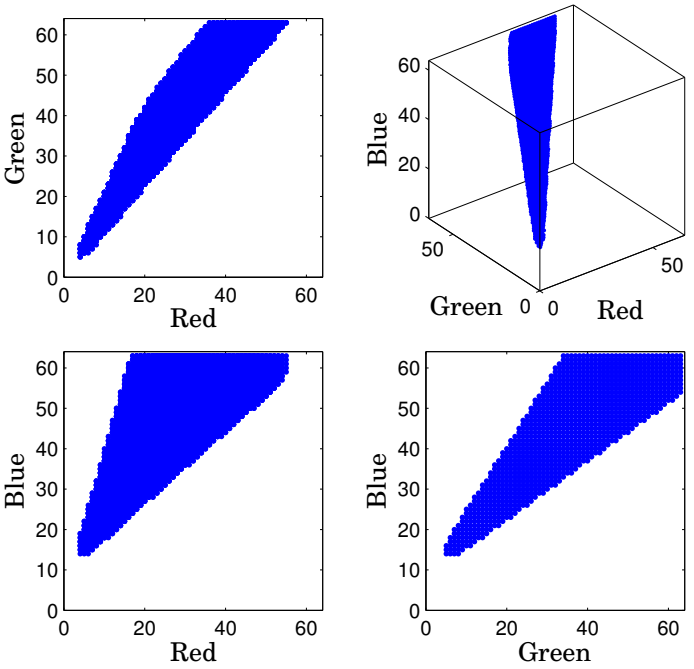


Figure 4.3 Visualization of the 3D look-up table used to obtain P_{fg} as a function of the color, viewed from four different directions. The diagrams show the elements that have a higher probability of belonging to the foreground than to the background.

were of the type Basler A602fc, capturing RGB images with 656×480 pixels. The cameras were running at 50 FPS (frames per second) with 4 ms exposure time. The thrown balls were usually moving so fast that the projections of a ball in two consecutive images were rarely overlapping. Hence, the image captured just before the one being analyzed could be used as background. When (4.8) was solved, 16 evenly spaced values of r were evaluated to see which gave the largest value of the maximand.

A look-up table with 64×64×64 elements, generated by the method described in Chapter 3, was used to calculate P_{fg} as a function of the intensities of red, green, and blue of a pixel. The values were, hence, in the range $0 \leq P_{fg} \leq 1$, and the elements with $P_{fg} > 0.5$ are marked with blue dots in Fig. 4.3. These elements represent a range of greenish chromaticities for a large range of intensities, forming a convex cone in the RGB space.

Detection of a Flying Ball

An example image, showing a green flying ball, can be seen in Fig. 4.4(a). The ball is in the lower right corner of the image. It was thrown by the person in the image and was flying toward a point beneath the camera. Motion blur in the vertical direction is visible. The region around the ball is magnified in Fig. 4.5(a).

In Fig. 4.4(b) and Fig. 4.5(b) the color of each pixel indicates its probability of belonging to the foreground, P_{fg}. The values were obtained from the look-up table shown in Fig. 4.3, assuming that there was no motion blur. It can be seen that the blurred upper and lower parts of the ball were classified to be background with high confidence, even though the pixels were exposed by the ball during a part of the exposure time.

To find the position of the ball, a score was calculated as $s = P_{fg} - 0.5$. Hence, s was positive if the pixel was most likely to belong to the foreground, and s was negative if the pixel was most likely to belong to the background. The ball was then located by finding the circle that locally maximized the sum of the scores of the pixels enclosed by the circle. This operation was performed efficiently using integral images [Viola and Jones, 2001].

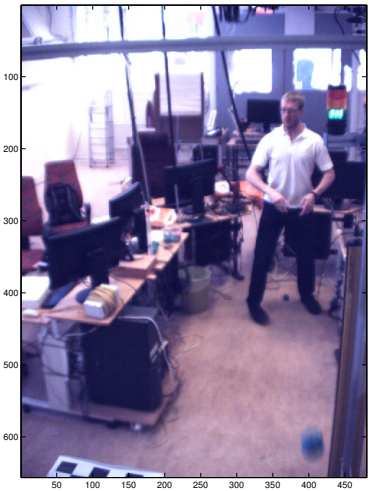
The result of the ball detection is shown as white circles in Figs. 4.5(a) and 4.5(b). The radius of the ball seems to be underestimated.

Figures 4.4(c) and 4.5(c) show R_{fg}, i.e., the pixel-wise probability that a foreground object was there during part of the exposure scaled by r according to (4.8). The region with $R_{fg} > 0.5$ is more circular than the region with $P_{fg} > 0.5$ in Fig. 4.5(b), and hence resembling the actual shape of the ball better. Above and below the circular red/yellow region in Fig. 4.5(c), there are cyan regions with R_{fg} significantly larger than zero, but less than 0.5, indicating a larger uncertainty of the position in the vertical direction than in the horizontal direction.

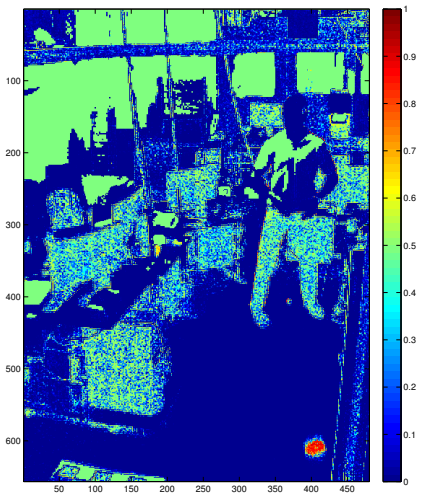
The result of the ball detection applied to R_{fg} is shown as black circles in Figs. 4.5(a) and 4.5(c). The estimated radius appears to be more accurate than for the white circle.

Figures 4.4(d) and 4.5(d) show the estimated value of r, i.e., the fraction of the exposure time that a pixel was exposed by a green ball. The estimation seems to have worked very well for the pixels that were actually exposed by the green ball. The value of r is close to 1 in the center of the ball, and r gradually decreases when going up or down in the image, toward the blurred regions.

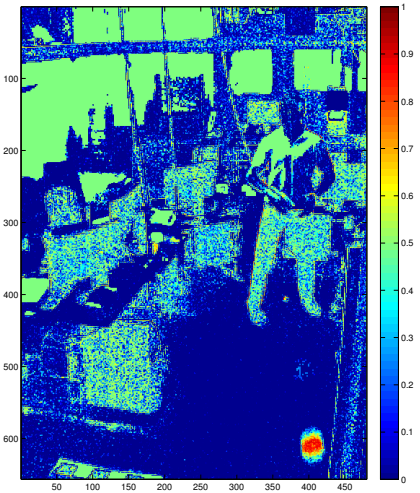
In the rest of the image, the estimates of r are basically only noise, but the effect on R_{fg} (which is the quantity used to detect foreground) is small, as expected from the analysis in Sec. 4.2. If you compare Figs. 4.4(b) and 4.4(c) you can see slightly more noise in R_{fg} than in P_{fg}, e.g., on the



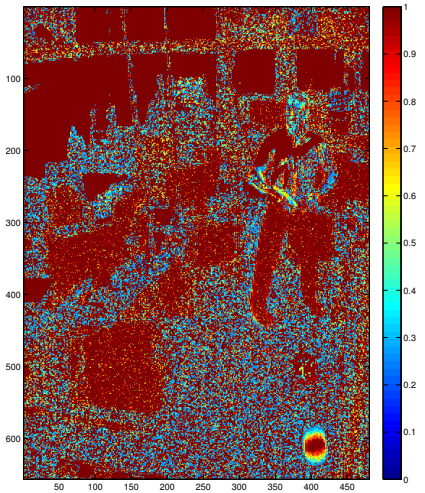
(a) RGB image.



(b) Probability of belonging to the foreground, $P_{fg}(c_{obs})$; without motion-blur estimation.

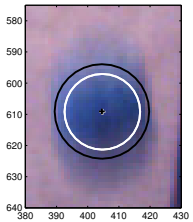


(c) R_{fg}, the probability that a foreground object passed the pixel during the exposure, scaled by $\hat{r}$; with motion-blur estimation.

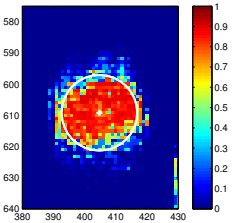


(d) Estimated values of r; the fraction of the exposure time that the pixels were exposed by the ball.

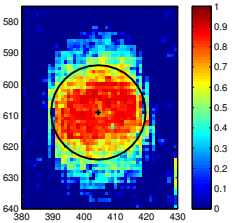
Figure 4.4 Analysis of an example image with a thrown green ball. The ball can be seen in the lower right corner of each image.



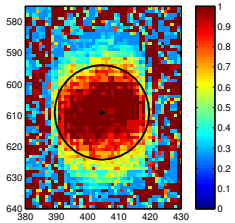
(a) RGB image.



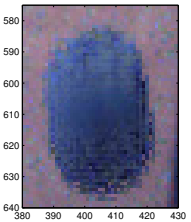
(b) $P_{fg}(c_{obs})$, without motion-blur estimation.



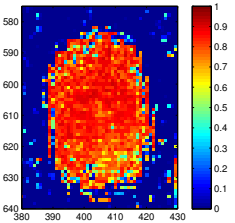
(c) R_{fg}, with motion-blur estimation.



(d) Estimates of r; $\hat{r}$.



(e) Estimates of the color of the moving object; $c_{mv}(\hat{r})$.



(f) $P_{fg}(c_{mv}(\hat{r}))$, the probability that a foreground object passed during the exposure.

Figure 4.5 Magnification of the region around the ball in Fig. 4.4. The circles show the estimated size and position of the ball. White circles: *without* motion-blur estimation. Black circles: *with* motion-blur estimation.

floor in the lower left corner.

On the edges of the person's right arm in Fig. 4.4(c), you can see large values of R_{fg}, caused by moving background objects as described on page 48. The values were, however, not large enough to cause erroneously detected foreground.

Improved Estimate of Size and Position

In order to evaluate how the motion-blur estimation affected the accuracy of the estimated position and radius of the ball, images of 31 throws were captured, resulting in totally 877 images of the ball. The images were captured with or without people walking around in the background, and the scene was lit by sunlight, fluorescent light, or a combination of both light sources. The ball thrown had a radius of 30 mm. Since the ball was observed by two different cameras, and the motion was tracked, it was possible to estimate the position of the ball at each measurement. When the position was known, it was possible to calculate the expected radius of the ball in the image, giving a ground truth for the radius estimation.

The tracking of the ball was performed by a Kalman filter running

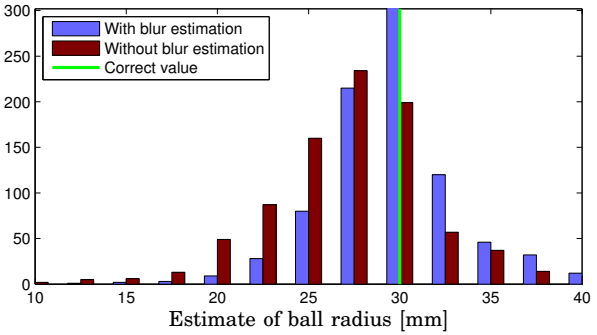


Figure 4.6 Histogram of the estimates of the ball radius. The correct radius, 30 mm, is indicated by a green line.

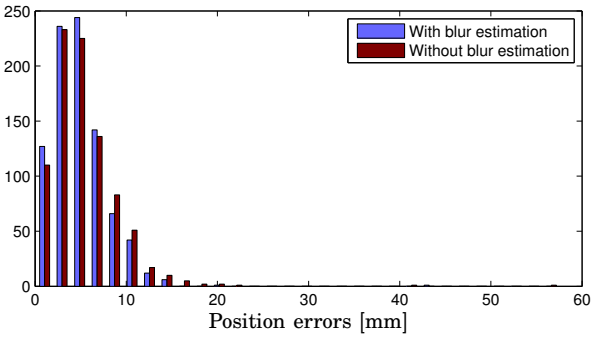


Figure 4.7 Histogram of the one-step prediction errors of the positions of the ball.

	All measurements		Bright background	
Blur est.	mean	std	mean	std
No	27.2 mm	4.7 mm	23.7 mm	3.2 mm
Yes	29.5 mm	4.0 mm	28.2 mm	1.6 mm

Table 4.1 Statistics on the performance of the ball-radius estimation, with and without motion-blur estimation. In the left half all measurements are included. The right half only includes measurements made with the bright yellow floor as background. The correct radius was 30 mm.

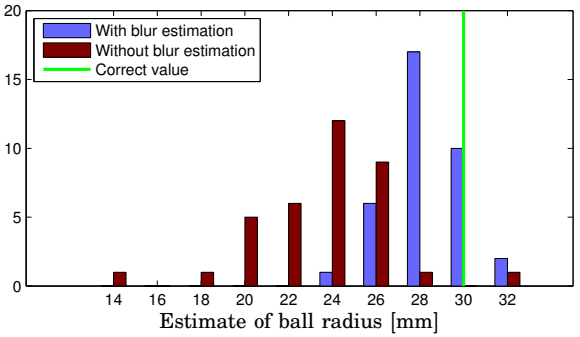


Figure 4.8 Histogram of the estimates of the ball radius. This figure only includes measurements that had the bright floor as background. The correct radius, 30 mm, is indicated by a green line.

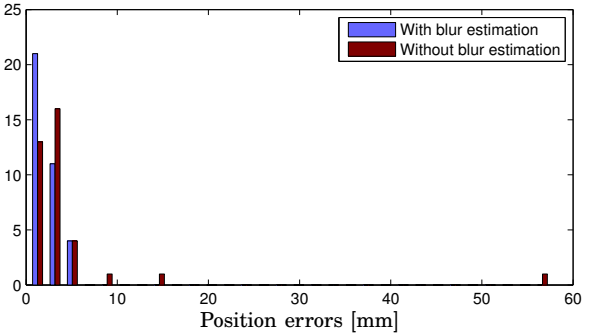


Figure 4.9 Histogram of the one-step prediction errors of the positions of the ball. This figure only includes measurements that were made with the bright floor as background.

with a time step of 20 ms, the same as the duration between two consecutive images. The ball was modeled as a point mass affected by gravity and air drag according to Eq. (2.4).

Figure 4.6 shows a histogram of the estimates of the ball radius, with and without motion-blur estimation. The mean and standard deviation are given in the left half of Table 4.1. It can be seen that the use of motion-blur estimation resulted in a mean value closer to the correct value, 30 mm, and a smaller standard deviation. In particular, the tail of underestimated radii was decreased.

The one-step prediction of the position of the ball was compared to the measured position, to evaluate the accuracy of the position estimates. The result, with and without motion-blur estimation, is displayed in Fig. 4.7,

showing that the motion-blur estimation gave a slightly reduced error. The mean error decreased from 5.4 mm to 4.9 mm.

When the ball was moving in front of a dark background, the colors in the blurred edges of the ball were very similar to the actual color of the ball, only darker. Hence, the radius estimation worked quite well with dark backgrounds, even without motion-blur estimation. When the background was the bright yellow floor, however, the colors in the blurred regions were very different from the green color of the ball, and then the use of the motion-blur estimation gave a very large improvement of the performance, as shown in Figs. 4.8 and 4.9, and the right half of Table 4.1. Figure 4.9 shows that the motion-blur compensation managed to recover an outlier position measurement.

Real-Time Execution

The motion-blur estimation was performed in real time to track balls for the ball-catching robot. The code was written in C and executed on one of the cores of an Intel® Core™2 Quad CPU Q6600 processor running at 2.40 GHz.

The approximate positions and radii of the balls were first detected using P_{fg}. In a square around each estimate, R_{fg} was then calculated and used to refine the estimates of the positions and radii.

The total execution time was approximately 5 ms per image, and the motion-blur estimation added 60 μs, i.e., an increase of only 1.2 %. The image analysis had to be performed on one pair of images every 20 ms, and the total time for solving the correspondence problem and executing the tracker was approximately 150 μs.

4.4 Discussion

In this chapter r was estimated for a single pixel at a time, giving benefits in the form of low complexity and fast computation time. To my knowledge, no previous method can estimate the motion blur for pixels individually. The estimates showed good accuracy where motion blur occurred, and the accuracy could probably be improved by estimating r jointly with the position and velocity of the ball, but it would come with the cost of higher complexity and longer computation times.

The motion-tracking methods [Dai et al., 2006; Perez et al., 2002; Shen et al., 2010; Wu et al., 2009; Wu et al., 2011] are more general than the method described in this chapter, but they have typically been reported to run at $10-20$ FPS on 320×240 pixel images. In terms of computation time, my method could run at 200 FPS on 656×480 pixel images.

Additionally, my method can run independently from the tracker, and does not need any prior information on where in the image the object should be. If the predictions from the tracker were used to limit the region to search for the ball in, the execution times reported in Sec. 4.3 could probably be reduced even more.

One drawback of the proposed method is that R_{fg} has slightly higher noise level than P_{fg}. For an erroneous large value of R_{fg} to occur, however, the ray extending from c_{bg} through c_{obs} has to intersect the foreground colors, combined with c_{obs} being closer to the foreground colors than to c_{bg}. In the experiments, the increased noise level had a small impact, compared to the improved performance in the regions where motion blur occurred.

One way to reduce the amount of false positives caused by the increased noise level, is to find the approximate position of the ball using P_{fg}, and then locally refine the position estimate using R_{fg}. For the ball-catching robot, false positives can also be filtered out if they do not match the motion model of a flying ball.

The proposed method does not model sensor noise, and it is argued why this only has a small impact on R_{fg}. A natural extension would be to model the noise, which probably could give better estimates of r, and possibly also better estimates of R_{fg}.

The discrepancy between the one-step predictions and the corresponding measurements, presented in the experimental results, were not caused only by image measurement errors, but also by prediction errors caused by load disturbances and modeling errors. Hence, the actual measurement errors were probably smaller than the values presented in Fig. 4.7.

Figure 4.5(e) shows the value of $c_{mv}(\hat{r})$, i.e., the estimated color of the object that caused motion blur in Figure 4.5(a). Fig. 4.5(f) shows the values of $P_{fg}(c_{mv}(\hat{r}))$, i.e., the estimated probability that the moving object was a foreground object. Figures 4.5(e) and 4.5(f) indicate which pixels were exposed by the ball during any part of the exposure time. The length of this region could be used to estimate how far the ball traveled during the exposure time and give an estimate of the velocity of the ball.

$P_{fg}(c_{mv}(\hat{r}))$ in Fig. 4.5(f) indicates which pixels were exposed by the ball during any part of the exposure, while $P_{fg}(c_{obs})$ in Fig. 4.5(b) indicates which pixels were exposed by the ball during the entire exposure. R_{fg} in Fig. 4.5(c) can be interpreted as an interpolation of $P_{fg}(c_{mv}(\hat{r}))$ and $P_{fg}(c_{obs})$, and can be used to calculate the average position of a foreground object during the exposure.

In the examples given, the previous image was used as background to the current image. It had the advantage that the background image adapted very quickly if the background or illumination changed, and this approach was possible since the foreground object was moving fast.

4.5 Conclusions

A method for handling motion blur in color-based detection was described. It estimates which pixels were subject to motion blur, as well as how large part of the exposure time the pixel was exposed by a foreground object. The method was successfully evaluated on data from a ball-catching robot system. The method was shown to increase the accuracy when measuring the positions and radii of flying balls. The method also executed fast, making it suitable for high-speed tracking applications.

5

Initialization of the Kalman Filter without Assumptions on the Initial State

5.1 Introduction

When performing state estimation on dynamical systems, the Kalman filter is a very commonly used tool [Kalman, 1960]. Just as for other recursive algorithms, initialization is necessary. In the original formulation of the Kalman filter, it was assumed that the distribution of the initial value of the state had a known mean value and known, finite covariance. If no such data are available, the estimate will have a transient in the initial phase of the filtering. If it is possible to start the estimation well before the estimate is to be used, this causes no problem, since the estimate will have time to converge. The transient can also be reduced by assuming that the initial value has a large covariance.

However, if the estimate is needed as soon as possible after the start of the estimation, it is desirable that not even the first estimates are biased by the guess of the initial state. One such example is a ball-catching robot, described further in Chapters 2 and 8. Only a limited number of measurements are available during the flight of the ball, and to make the motion of the robot as smooth as possible, it has to start moving as soon as the catching position can be estimated. Thus, it is essential to have a good estimate from the very start of the measurement series so the robot can get to its target in time.

The class of systems considered in this chapter is discrete-time time-varying linear systems on the form

$$\begin{aligned} x(k+1) &= \Phi(k)x(k) + \Gamma(k)u(k) + v(k) \\ y(k) &= C(k)x(k) + e(k) \end{aligned} \tag{5.1}$$

where $x(k)$ is the state vector, $u(k)$ is an input vector and $y(k)$ is a measurement vector. The matrix $\Phi(k)$ describes the system dynamics, $\Gamma(k)$ describes how the system is affected by the input, and $C(k)$ describes the measurement function. The disturbances v and e are assumed to be white-noise processes with zero mean values, $\mathrm{E}[v(k)v^T(k)] = R_v(k) \succeq 0$, $\mathrm{E}[e(k)e^T(k)] = R_e(k) \succ 0$, and $\mathrm{E}[v(k)e^T(k)] = 0$. The number of states is assumed to be constant, but the numbers of inputs and outputs can be time-varying.

Many specialized approaches for making an informed initialization of the Kalman filter have been proposed for specific problems. In [Linderoth, 2008] it was assumed that a flying ball with the model (2.5) was always observed by two cameras simultaneously, making it possible to measure the 3D position in every time step. When the ball had been observed at two different time instants, both position and velocity could then be estimated. In [Einhorn et al., 2007] a mobile robot was used to estimate the positions of stationary features. Taking images from two different positions and using the odometry measurements from the wheels, it was possible to estimate the positions of the features. These estimates were used as initial values and were incrementally improved by means of a Kalman filter. In [Hyland, 2002] and [Weiner, 1981] special care was taken when estimating the subspace that was observable from the first measurement. The estimation was then continued assuming large covariance in the remaining directions.

A more general way to initialize (5.1) is to stack all measurements to form a large system of linear equations:

$$\underbrace{\begin{bmatrix} y(0) \\ y(1) - C(1)\Gamma(0)u(0) \\ y(2) - C(2)\left[\Phi(1)\Gamma(0)u(0) + \Gamma(1)u(1)\right] \\ \vdots \end{bmatrix}}_{y'} = \underbrace{\begin{bmatrix} C(0) \\ C(1)\Phi(0) \\ C(2)\Phi(1)\Phi(0) \\ \vdots \end{bmatrix}}_{C'} x(0) + e' \tag{5.2}$$

where all the disturbances are collected in e'. The value of $x(0)$ can be estimated if C' gets full row rank. One problem with (5.2) is that y' and C' may become very large if many measurements are needed before C' gets full row rank. One must also take into consideration how $y(k)$ are correlated for all k.

An alternative approach is to use the information form of the Kalman filter [Kailath et al., 2000]. In the standard Kalman filter, the information is represented by the state estimate, $\hat{x}$, and its covariance matrix, P. In the information form of the filter, the state is represented by the information matrix, $Z = P^{-1}$, and the information vector, $\hat{z} = P^{-1}\hat{x}$. The

fact that nothing is known about the state can then simply be represented as $Z = 0$, corresponding to infinite covariance. However, the time update of the information filter puts extra requirements on the system (5.1), e.g., that Φ is non-singular or that (5.1) is controllable.

A general method for initializing (5.1) with infinite initial covariance was presented in [Hagander, 1973]. It maintains a matrix $\Lambda(k)$ that spans the directions that have infinite variance, which is used to determine an alternative measurement update equation for the Kalman filter. $\Lambda(k)$ looses rank when more measurements are introduced and becomes the zero matrix when the state becomes observable. In the directions with infinite variance, all states are equally likely. The state estimate is represented by one out of these infinitely many points.

This chapter presents an alternative general way of initializing (5.1) with infinite initial covariance. This is done by transforming the state estimate into a basis where the directions with infinite variance are spanned by a subset of the basis vectors. In the subspace spanned by the remaining basis vectors, the variance is finite.

5.2 Preliminaries

The Kalman Filter

The Kalman filter can be used to estimate the state of (5.1) recursively as described by

$$\begin{aligned}
\hat{x}(k|k) &= \hat{x}(k|k-1) + K(k)\left(y(k) - C(k)\hat{x}(k|k-1)\right) \\
K(k) &= P(k|k-1)C^T(k)\left(C(k)P(k|k-1)C^T(k) + R_e(k)\right)^{-1} \\
P(k|k) &= P(k|k-1) - K(k)C(k)P(k|k-1) \\
\hat{x}(k+1|k) &= \Phi(k)\hat{x}(k|k) + \Gamma(k)u(k) \\
P(k+1|k) &= \Phi(k)P(k|k)\Phi^T(k) + R_v(k)
\end{aligned} \tag{5.3}$$

where $\hat{x}(l|k)$ denotes the estimate of $x(l)$ based on measurements up to sample k, and $P(l|k)$ denotes the covariance matrix of $\hat{x}(l|k)$.

Singular Value Decomposition

Consider a matrix $A \in \mathbb{R}^{m\times n}$ with $\text{rank}(A) = r$. Using singular value decomposition (SVD) it can be factorized as

$$A = U\Sigma V^T \tag{5.4}$$

where $U \in \mathbb{R}^{m\times r}$ satisfies $U^TU = I$, $V \in \mathbb{R}^{n\times r}$ satisfies $V^TV = I$ and $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$ with $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$

5.3 Optimal Solution of a Linear System of Equations with Noise

This section describes methods for solving linear systems of equations, which will be used later in this chapter.

Over-Determined System

Consider a system of linear equations with disturbances:

$$z = Gx + w \tag{5.5}$$

where $z \in \mathbb{R}^m$ and $G \in \mathbb{R}^{m \times n}$ are known and $w \in \mathbb{R}^m$ is a disturbance with $\mathrm{E}[w] = 0$ and $\mathrm{E}[ww^T] = R_w \succ 0$.

Assume that $\mathrm{rank}(G) = n < m$, i.e., the system is over-determined. Let $\hat{x}$ denote the minimum-variance unbiased estimate of x, which is given by

$$\hat{x} = (G^T R_w^{-1} G)^{-1} G^T R_w^{-1} z \tag{5.6}$$

$$R_x = \mathrm{E}\left[(\hat{x} - x)(\hat{x} - x)^T\right] = (G^T R_w^{-1} G)^{-1} \tag{5.7}$$

according to the Gauss-Markov theorem [Kailath et al., 2000].

Under-Determined System

Again consider the system (5.5), but now assume that $\mathrm{rank}(G) = r < n$, i.e., the system of equations is under-determined. Still, x can be partly determined. By singular value decomposition G can be factorized as

$$G = U \Sigma V^T \tag{5.8}$$

It is possible, as a part of the SVD algorithm, to construct

$$S = \begin{bmatrix} S_f & S_i \end{bmatrix} \in \mathbb{R}^{n \times n} \tag{5.9}$$

such that $S_f = V$ and $S^T S = I$. Define $x_f \in \mathbb{R}^r$ and $x_i \in \mathbb{R}^{n-r}$ as the unique solution to

$$x = S \begin{bmatrix} x_f \\ x_i \end{bmatrix} = S_f x_f + S_i x_i \tag{5.10}$$

Note that x_f is a parameterization of the part of x that can be estimated by (5.5), and x_i is a parameterization of the null space of G. Inserting (5.8) and (5.10) into (5.5) and noting that $V^T [S_f \; S_i] = [I \; 0]$ one obtains

$$\begin{aligned} z &= Gx + w \\ &= U \Sigma V^T (S_f x_f + S_i x_i) + w \\ &= U \Sigma x_f + w \end{aligned} \tag{5.11}$$

Since $U\Sigma$ has full row rank, one can solve (5.11) for $\hat{x}_f$, using the method for over-determined systems described in the previous subsection.

5.4 Filter Initialization

State Partitioning

During the initialization of the Kalman filter there may be times when the variance of the state estimate is finite in some directions of the state space and infinite in other directions. To handle this situation the state can by a linear transformation be reoriented to a space where the directions with infinite variance are orthogonal to as many basis vectors as possible. Let the state $\bar{x}$ in this alternative space be defined by

$$T\bar{x} = x \tag{5.12}$$

where $T \in \mathbb{R}^{n\times n}$ and $T^T T = I$. Let $\hat{\bar{x}}$ denote the estimate of $\bar{x}$, and denote the estimation error by

$$\tilde{\bar{x}} = \bar{x} - \hat{\bar{x}} \tag{5.13}$$

Throughout the chapter it is assumed that the estimates are designed to be unbiased, i.e., so that $\mathrm{E}\,[\tilde{\bar{x}}] = 0$. The state can be partitioned as

$$\bar{x} = \begin{bmatrix} \bar{x}_f \\ \bar{x}_i \end{bmatrix} \tag{5.14}$$

such that the covariance-matrix of $\hat{\bar{x}}_f$ is finite and the variance of $\hat{\bar{x}}_i$ is infinite in all directions. Define n_f and n_i such that

$$n_f + n_i = n \tag{5.15}$$

and $\bar{x}_f \in \mathbb{R}^{n_f}$, $\bar{x}_i \in \mathbb{R}^{n_i}$. Similarly, the transformation matrix T can be partitioned as

$$T = \begin{bmatrix} T_f & T_i \end{bmatrix}, \quad T_f \in \mathbb{R}^{n\times n_f}, \quad T_i \in \mathbb{R}^{n\times n_i} \tag{5.16}$$

Since T is orthonormal we have

$$T^{-1} = \begin{bmatrix} T_f & T_i \end{bmatrix}^{-1} = \begin{bmatrix} T_f & T_i \end{bmatrix}^T = \begin{bmatrix} T_f^T \\ T_i^T \end{bmatrix} \tag{5.17}$$

Note for future reference that inserting (5.14) and (5.17) into (5.12) gives

$$\begin{bmatrix} \bar{x}_f \\ \bar{x}_i \end{bmatrix} = \begin{bmatrix} T_f^T \\ T_i^T \end{bmatrix} x \tag{5.18}$$

and inserting (5.16) into (5.12) gives

$$x = T_f \bar{x}_f + T_i \bar{x}_i \tag{5.19}$$

which shows that T_f spans the directions in which $\hat{x}$ has finite variance and T_i spans the directions in which $\hat{x}$ has infinite variance.

Let $\bar{P}_f$ denote the covariance matrix of $\hat{\bar{x}}_f$:

$$\bar{P}_f = \mathrm{E}\left[(\hat{\bar{x}}_f - \bar{x}_f)(\hat{\bar{x}}_f - \bar{x}_f)^T\right] \tag{5.20}$$

In the remainder of the chapter all quantities may be appended with time indices so that, for example, $\hat{\bar{x}}_f(l|k)$ is the estimate of $\bar{x}_f(l)$ based on measurements up to sample k, and $\hat{x}(l|k) = T(l|k)\hat{\bar{x}}(l|k)$. Note, however, the slightly different case $x(l) = T(l|k)\bar{x}(l|k)$. The actual state x has only a single time index, since the second time index is meaningful only for estimates. Still $\bar{x}$ has two time indices to indicate which T was used for the transformation.

To conclude, all the knowledge about $\hat{x}(l|k)$ can be fully specified by $T(l|k)$, $\hat{\bar{x}}_f(l|k)$ and $\bar{P}_f(l|k)$.

Time Update

Assume that $T(k|k)$, $\hat{\bar{x}}_f(k|k)$ and $\bar{P}_f(k|k)$ are known. The purpose of the time-update operation is to calculate $T(k+1|k)$, $\hat{\bar{x}}_f(k+1|k)$, and $\bar{P}_f(k+1|k)$, i.e., an estimate of $x(k+1)$ based on measurements up to sample k. The time-update equation of the state model (5.1) is given by

$$x(k+1) = \Phi(k)x(k) + \Gamma(k)u(k) + v(k) \tag{5.21}$$

but it can not be used for calculations directly if $\hat{x}(k)$ does not have finite variance.

Choose the transformation to be used for the state estimate after the time update, $T(k+1|k)$, such that

$$T_f^T(k+1|k)\Phi(k)T_i(k|k) = 0 \tag{5.22}$$

$$\begin{aligned} n_f(k+1|k) &= \mathrm{rank}(T_f(k+1|k)) \\ &= n - \mathrm{rank}\left(\Phi(k)T_i(k|k)\right) \end{aligned} \tag{5.23}$$

$$T^T(k+1|k)T(k+1|k) = I \tag{5.24}$$

Equation (5.22) says that the columns of $T_f(k+1|k)$ should be in the left null space of $\Phi(k)T_i(k|k)$. Why this is a good idea will become clear in Eq. (5.28). The condition (5.23) is imposed to make sure that $T_f(k+1|k)$ spans the *entire* left null space of $\Phi(k)T_i(k|k)$. Equation (5.24) makes sure that the transformation is orthonormal. The calculation of $T_f(k+1|k)$ can be done by means of SVD.

Note that

$$\mathrm{rank}\left(\Phi(k)T_i(k|k)\right) \leq \mathrm{rank}\left(T_i(k|k)\right) \tag{5.25}$$

Inserting (5.25) and (5.15) into (5.23) gives

$$\begin{aligned} n_f(k+1|k) &= n - \operatorname{rank}\left(\Phi(k)T_i(k|k)\right) \\ &\geq n - \operatorname{rank}\left(T_i(k|k)\right) \\ &= n - n_i(k|k) \\ &= n_f(k|k) \end{aligned} \tag{5.26}$$

The conclusion of (5.26) is that

$$n_f(k+1|k) \geq n_f(k|k) \tag{5.27}$$

where strict inequality holds if and only if $\Phi(k)$ is singular and its null space satisfies $\mathcal{N}(\Phi(k)) \cap \mathcal{R}(T_i(k|k)) \neq \varnothing$.

Now that we have chosen a state transformation, we go on to see how it affects the state estimation. Premultiplying (5.21) with $T_f^T(k+1|k)$ gives

$$\begin{aligned} &\bar{x}_f(k+1|k) \\ &= T_f^T(k+1|k)x(k+1) \\ &= T_f^T(k+1|k)\Phi(k)x(k) \\ &\quad + T_f^T(k+1|k)\Gamma(k)u(k) \\ &\quad + T_f^T(k+1|k)v(k) \\ &= T_f^T(k+1|k)\Phi(k)\left(T_f(k|k)\bar{x}_f(k|k) + T_i(k|k)\bar{x}_i(k|k)\right) \\ &\quad + T_f^T(k+1|k)\Gamma(k)u(k) \\ &\quad + T_f^T(k+1|k)v(k) \\ &= T_f^T(k+1|k)\Phi(k)T_f(k|k)\bar{x}_f(k|k) \\ &\quad + T_f^T(k+1|k)\Gamma(k)u(k) \\ &\quad + T_f^T(k+1|k)v(k) \end{aligned} \tag{5.28}$$

where the equalities result from (5.18), (5.21), (5.19), and (5.22), respectively. Here the advantage of choosing $T(k+1|k)$ according to (5.22) becomes clear. Because of this choice $\bar{x}_f(k+1)$ is independent of $\bar{x}_i(k)$ and only depends on quantities with finite variance. Condition (5.23) guarantees that $T_f(k+1)$ has the highest possible rank.

Motivated by (5.28), let the time update of the state estimate be defined by

$$\begin{aligned} \hat{\bar{x}}_f(k+1|k) = {} & T_f^T(k+1|k)\Phi(k)T_f(k|k)\hat{\bar{x}}_f(k|k) \\ & + T_f^T(k+1|k)\Gamma(k)u(k) \end{aligned} \tag{5.29}$$

Inserting (5.28) and (5.29) into (5.13) gives

$$\begin{aligned}\tilde{\tilde{x}}_f(k+1|k) &= \bar{x}_f(k+1|k) - \hat{\tilde{x}}_f(k+1|k) \\ &= T_f^T(k+1|k)\Phi(k)T_f(k|k)\tilde{\tilde{x}}_f(k|k) \\ &\quad + T_f^T(k+1|k)v(k)\end{aligned} \tag{5.30}$$

It is easily verified that $\mathrm{E}\,[\tilde{\tilde{x}}_f(k+1|k)] = 0$ as required. The variance of the estimate becomes

$$\begin{aligned}\bar{P}_f(k+1|k) &= \mathrm{E}\left[\tilde{\tilde{x}}_f(k+1|k)\tilde{\tilde{x}}_f^T(k+1|k)\right] \\ &= Q\bar{P}_f(k|k)Q^T + T_f^T(k+1|k)R_v(k)T_f(k+1|k),\end{aligned} \tag{5.31}$$

where $Q = T_f^T(k+1|k)\Phi(k)T_f(k|k)$

Measurement Update

Assume that $T(k|k-1)$, $\hat{\tilde{x}}_f(k|k-1)$ and $\bar{P}_f(k|k-1)$ are known. The purpose of the measurement update is to calculate $T(k|k)$, $\hat{\tilde{x}}_f(k|k)$ and $\bar{P}_f(k|k)$. The state model (5.1) gives the measurement equation

$$y(k) = C(k)x(k) + e(k) \tag{5.32}$$

Combining (5.13) and (5.18) gives

$$\begin{aligned}\hat{\tilde{x}}_f(k|k-1) &= \bar{x}_f(k|k-1) - \tilde{\tilde{x}}_f(k|k-1) \\ &= T_f^T(k|k-1)x(k) - \tilde{\tilde{x}}_f(k|k-1)\end{aligned} \tag{5.33}$$

where the last term is a zero-mean Gaussian random variable with covariance given by (5.31).

Equations (5.32) and (5.33), containing the information from the new measurement and from the previous state estimate, respectively, can be formulated as a single linear system of equations:

$$\underbrace{\begin{bmatrix} y(k) \\ \hat{\tilde{x}}_f(k|k-1) \end{bmatrix}}_{z} = \underbrace{\begin{bmatrix} C(k) \\ T_f^T(k|k-1) \end{bmatrix}}_{G} x(k) + \underbrace{\begin{bmatrix} e(k) \\ -\tilde{\tilde{x}}_f(k|k-1) \end{bmatrix}}_{w} \tag{5.34}$$

which can be solved by the method for under-determined systems, described on page 61, with

$$R_w = \begin{bmatrix} R_e(k) & 0 \\ 0 & \bar{P}_f(k|k-1) \end{bmatrix} \tag{5.35}$$

The solution is given by

$$T(k|k) = S \tag{5.36}$$

$$\hat{\hat{x}}_f(k|k) = (\Sigma U^T R_w^{-1} U \Sigma)^{-1} \Sigma U^T R_w^{-1} z \tag{5.37}$$

$$\bar{P}_f(k|k) = (\Sigma U^T R_w^{-1} U \Sigma)^{-1} \tag{5.38}$$

$$n_f(k|k) = \text{rank}(\Sigma) = \text{rank}(G) \tag{5.39}$$

where U, Σ and S are defined in (5.8) and (5.9).

From the definition of G in (5.34) it can be seen that

$$\text{rank}(G) \geq \text{rank}\,(T_f(k|k-1)) \tag{5.40}$$

The orthonormality of $T(k|k-1)$ in combination with (5.16) gives

$$\text{rank}\,(T_f(k|k-1)) = n_f(k|k-1) \tag{5.41}$$

Combining Eqs. (5.39)–(5.41) results in

$$n_f(k|k) \geq n_f(k|k-1) \tag{5.42}$$

where equality holds if and only if $\mathcal{R}(C^T(k)) \subseteq \mathcal{R}(T_f(k|k-1))$. Equations (5.27) and (5.42) together show that n_f (the dimension of the subspace with finite variance) never decreases and give conditions for when n_f increases.

If G has full rank the variance of $\hat{x}(k|k)$ will be finite in all directions and $n_f(k|k) = n$.

Remark: For $n_f(k|k-1) = n$ and $T_f(k|k-1) = I$ it can be shown that the solution of (5.34) is equivalent to the measurement update of the ordinary Kalman filter (5.3).

How to Start and When to Stop

Assuming that nothing is known about x when the estimation starts out ($n_f = 0$), the first thing to do is to apply the measurement update to the first measurement. The lower blocks of the matrices z and G, and all blocks except R_e in R_w, will then be empty, since there is no prior state estimate.

If the initial variance of $\hat{x}$ is infinite only in some directions ($0 < n_f < n$) and the available information can be represented by a system of linear equations with noise, then this prior information can be plugged into Eq. (5.34) directly.

If the measurements provide enough information, the state becomes observable and the variance of the estimate will be finite in all directions ($n_i = 0$) after a number of iterations of the filter. Then, it is no longer necessary to use the algorithm described in this chapter, and one can just as well use the standard Kalman filter (5.3), since the methods are equivalent for $n_i = 0$.

5.5 Simulation Example

To illustrate the use of the filter, consider a flying ball affected by gravity and negligible air drag. The ball is tracked by a vision system, where each camera can provide an estimate of the line that intersects both the ball and the focal point of the camera, but no depth information is available. The process model is given by Eq. (2.5) on page 24 with the white-noise load-disturbance covariance $R_v = 10^{-6} I_{6\times 6}$. To make the example easy to follow, we use the sampling period $h = 1$ and the Earth gravity constant $g = 10$. Let the initial state of the system be $x(0) = [1\ \ 2\ \ 3\ \ 0\ \ 1\ \ 4]^T$, where the first three components are positions and the last three components are velocities. The trajectory of the ball is shown as a black curve in Fig. 5.1. The positions of the ball at the measuring instants are marked with green circles and the corresponding lines that are extracted from the images are marked in red. One camera observes the ball at time steps 0 and 2, and another camera observes the ball at time step 1. The simulated measurements are given by

$$y(0) = \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \qquad C(0) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{5.43}$$

$$y(1) = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \qquad C(1) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.44}$$

$$y(2) = \begin{bmatrix} -1.9 \\ 1 \end{bmatrix}, \qquad C(2) = \begin{bmatrix} 0.4 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{5.45}$$

$$R_e(0) = R_e(1) = R_e(2) = 10^{-4} I_{2\times 2} \tag{5.46}$$

Performing the state estimation on the given data gives the following results:

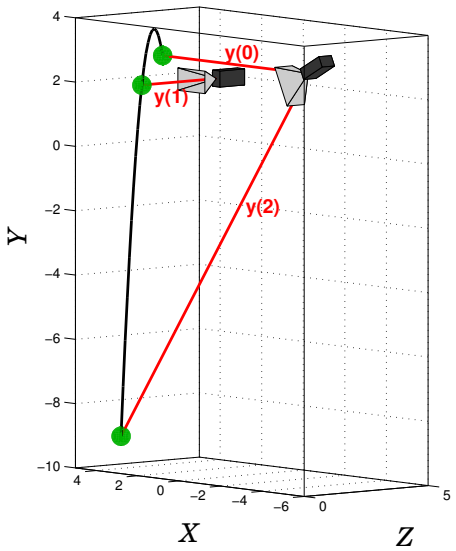


Figure 5.1 Simulated ball trajectory and measurements marked as green balls.

Estimates at time 0 based on measurements up to time 0.

$$T_f(0|0) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.47}$$

$$T_i(0|0) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.48}$$

$$\hat{\hat{x}}_f(0|0) = \begin{bmatrix} 3 \\ 1 \end{bmatrix} \tag{5.49}$$

$$\bar{P}_f(0|0) = 10^{-4} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.50}$$

Estimates at time 1 based on measurements up to time 0.

$$T_f(1|0) = \begin{bmatrix} 0 & 0 \\ 0.707 & 0 \\ 0 & 0.707 \\ 0 & 0 \\ -0.707 & 0 \\ 0 & -0.707 \end{bmatrix} \tag{5.51}$$

$$T_i(1|0) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.707 & 0 & 0 \\ 0 & 0 & 0 & 0.707 \\ 0 & 0 & 1 & 0 \\ 0 & 0.707 & 0 & 0 \\ 0 & 0 & 0 & 0.707 \end{bmatrix} \tag{5.52}$$

$$\hat{\hat{x}}_f(1|0) = \begin{bmatrix} 5.657 \\ 0.707 \end{bmatrix} \tag{5.53}$$

$$\bar{P}_f(1|0) = 10^{-4} \begin{bmatrix} 0.51 & 0 \\ 0 & 0.51 \end{bmatrix} \tag{5.54}$$

Estimates at time 1 based on measurements up to time 1.

$$T_f(1|1) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0.707 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -0.707 & 0 & 0 & 0 \end{bmatrix} \tag{5.55}$$

$$T_i(1|1) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0.707 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0.707 \end{bmatrix} \tag{5.56}$$

$$\hat{\hat{x}}_f(1|1) = \begin{bmatrix} 0.707 \\ 3 \\ 2 \\ -6 \end{bmatrix} \tag{5.57}$$

$$\bar{P}_f(1|1) = 10^{-4} \begin{bmatrix} 0.51 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 2.02 \end{bmatrix} \tag{5.58}$$

Estimates at time 2 based on measurements up to time 1.

$$T_f(2|1) = \begin{bmatrix} 0 & 0.707 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0.447 & 0 & 0 & 0 \\ 0 & -0.707 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -0.894 & 0 & 0 & 0 \end{bmatrix} \tag{5.59}$$

$$T_i(2|1) = \begin{bmatrix} 0.707 & 0 \\ 0 & 0 \\ 0 & 0.894 \\ 0.707 & 0 \\ 0 & 0 \\ 0 & 0.447 \end{bmatrix} \tag{5.60}$$

$$\hat{\hat{x}}_f(2|1) = \begin{bmatrix} 0.447 \\ 2.121 \\ -9 \\ -16 \end{bmatrix} \tag{5.61}$$

$$\bar{P}_f(2|1) = 10^{-4} \begin{bmatrix} 0.214 & 0 & 0 & 0 \\ 0 & 0.51 & 0 & 0 \\ 0 & 0 & 5.03 & 3.02 \\ 0 & 0 & 3.02 & 2.03 \end{bmatrix} \tag{5.62}$$

Estimates at time 2 based on measurements up to time 2.

$$T_f(2|2) = I_{6\times 6} \tag{5.63}$$

$$T_i(2|2) \in \mathbb{R}^{6\times 0} \tag{5.64}$$

$$\hat{\hat{x}}_f(2|2) = \begin{bmatrix} 2 \\ -9 \\ 1 \\ -1 \\ -16 \\ 0 \end{bmatrix} \tag{5.65}$$

$$\bar{P}_f(2|2) = 10^{-4} \begin{bmatrix} 9.08 & -3.77 & 0 & 9.08 & -2.27 & 0 \\ -3.77 & 5.03 & 0 & -3.77 & 3.02 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0.5 \\ 9.08 & -3.77 & 0 & 10.1 & -2.27 & 0 \\ -2.27 & 3.02 & 0 & -2.27 & 2.03 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0.52 \end{bmatrix} \tag{5.66}$$

Most insight on the estimation progress is given by studying T_f. The first measurement locates the ball on a line in the X-direction, which gives information about the position in the Y- and Z-directions. This is reflected in the columns of $T_f(0|0)$. After the time update the position is no longer known. Only linear combinations of the positions and velocities can be determined, as seen in $T_f(1|0)$. With the second measurement, the coordinates in the X- and Y-directions are given, as seen in $T_f(1|1)$. Since it is the second measurement in the Y-direction, $\dot{Y}$ can be determined. Still, no information about $\dot{X}$ is available and, hence, X is no longer known after the time update, as indicated by $T_f(2|1)$. The last measurement gives information in the directions that still have infinite variance, and thus $T_f(2|2)$ spans the entire $\mathbb{R}^n$ and an estimate $\hat{x}(2|2) = T_f(2|2)\hat{\bar{x}}(2|2)$ can finally be computed.

5.6 Discussion

The reason for doing the partitioning suggested in this chapter, is the difficulty of representing matrices with infinite singular values and states with correlated infinite variances. Special care must be taken before giving any numerical values of the state estimate $\hat{x}$ in the original state space. To see the complications, consider Eq. (5.19). If a row in T_i has any non-zero element, then the corresponding element of $\hat{x}$ has infinite variance. The method in [Hagander, 1973] calculated an estimate in the original state space and pointed out that is was one out of infinitely many, equally likely estimates within a subspace. The method presented in this thesis instead finds a transformation where $\bar{x}_i$ is a minimal parameterization of the state along the directions with infinite variance, and $\bar{x}_f$ is a minimal parameterization of the state along the directions with finite variance. The full knowledge about the state is represented by T, $\hat{\bar{x}}_f$ and $\bar{P}_f$.

Equations (5.22)–(5.24) and (5.34) do not in general have a unique solution for T. In this thesis SVD-based methods for solving the equations are suggested, but other methods can be used. The transformation T can be replaced by any T' fulfilling Eq. (5.12)–(5.20) such that $\mathcal{R}(T_f) = \mathcal{R}(T'_f)$. Of course $\hat{\bar{x}}_f$ and $\bar{P}_f$ have to be modified accordingly. In the example on pages 67–70, the T matrices were chosen to align the basis vectors of $\bar{x}_f$ with the basis vectors of the original state space as far as possible to improve human readability.

The presented initialization procedure is useful when very little is known about the initial state. If *a priori* knowledge is available, this should of course be used to improve the estimate.

The state $x = x(k)$, and hence also $\bar{x}_f$ and $\bar{x}_i$, are assumed to have

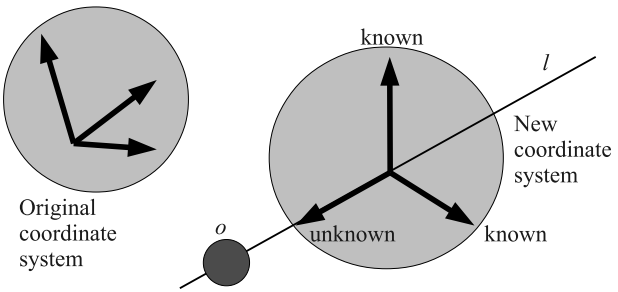


Figure 5.2 Example illustrating state partitioning. The position of the object o along the line l is unknown, and a new coordinate system, where as many basis vectors as possible are orthogonal to the line, is chosen.

exact and finite time-varying values, although not known exactly. More specifically it is assumed that no information at all is available about $\bar{x}_i$, which is modeled as $\hat{\bar{x}}_i$ having infinite variance.

The information in $\hat{\bar{x}}_f$ can be used to give partial information about the state before the full state is observable. For instance, it may be of interest to know the altitude of an aerial vehicle before its longitude and latitude can be estimated.

As an example of state partitioning, consider the scenario in Fig. 5.2, where an object o is known to be near a given line l in 3D-space, but nothing is known about its position along the line. Choose a coordinate system such that its first two basis vectors are orthogonal to l and the third basis vector is parallel to l. The position of o can then be partly described by the first two components with a finite covariance matrix, even though the variance in the direction of the third component (parallel to l) is infinite.

The method presented in this chapter can handle infinite initial variances, but in practice most (all?) real quantities have finite variance. The development of the method was motivated by the need to quickly get good estimates for the ball-catching robot, no matter where the ball was thrown from. However, considering the size of the room, the field of view and the resolution of the cameras, and the physical limitations of a human pitcher, it was safe to say the the distance from the robot to the ball was less than 15 m and the speed was less than 30 m/s. Using this information to form an initial estimate did not cause any numerical problems for modern processors, and it should generate more accurate estimates than a method that does not make use of the actually available prior information. In the light of these aspects, the initialization method presented here may be considered a theoretical contribution, rather than a practical one.

5.7 Conclusions

A new way of initializing the Kalman filter was presented, making it possible to calculate a state estimate that is not influenced by any guess of the initial value of the state. Instead, the estimate can be determined completely based on the first measurements.

6

Object Tracking with Measurements from Single or Multiple Cameras

6.1 Introduction

To be able to determine the position of a static object in 3D space by means of computer vision, the object has to be observed by cameras from at least two different view points, assuming that the size of the object is not known. The same applies for measuring the position of a moving object based on images captured at one single time instant. However, if the cameras do not capture images simultaneously, or if a moving object is not visible in all images, one can not rely on using triangulation methods for making accurate position estimates of dynamical objects.

This chapter describes a strategy for tracking an object with known dynamical model, using a series of images where no pair has to be captured simultaneously. It allows tracking of a point object in 3D space using a single static camera, and provides a convenient way of fusing data from multiple cameras.

The properties presented in the previous paragraph are very desirable for a ball-catching robot. They allow any number of cameras to be used, though at least two cameras are needed for good accuracy if the cameras are static. If the position of the ball can not be extracted from an image, due to occlusions or failed image analysis, the data from other images captured simultaneously can still be used to improve the estimate of the state of the ball. No explicit triangulation is needed.

An overview of tracking techniques is given in [Yilmaz et al., 2006]. Many methods focus on recognizing objects and determining correspondences between frames. The recognition can be done using template matching [Schweitzer et al., 2002], color histograms [Birchfield, 1998] in

combination with the mean-shift algorithm [Comaniciu and Meer, 1999], SIFT features [Lowe, 1999], etc. When tracking a set of unlabeled points, different approaches exist for determining correspondences. One possibility is to try to match a point in an image with the point being closest to it in the next image [Salari and Sethi, 1990]. The method can be improved by estimating the velocities of the points and assuming near-zero acceleration. A method presented in [Ding et al., 2007] formed a Hankel matrix out of the sequence of image point coordinates and analyzed the rank of this matrix to identify tracklets obeying the same dynamical model. A similar approach was used in [Lublinerman et al., 2006] to partition image points into groups belonging to the same rigid object.

For a ball-catching robot it is not enough to determine correspondences between image frames. It is also necessary to accurately determine the state of the ball and perform reliable predictions of the future motion. The method of this chapter does so by means of a Kalman filter [Kalman, 1960] and a dynamical 3D model. A compact way of relating image coordinates to 3D rays and performing outlier detection is presented. The outlier detection threshold is based on the uncertainties of the measurement and the state estimate. The method was first published in [Linderoth et al., 2010]

A similar method for parameterizing the ray relating to an image point was presented in [Lippiello and Ruggiero, 2012a]. It was used to catch balls with a single mobile camera. The state estimation was done by simulating the ball trajectory and trying to find the initial position and velocity that best matched the measurements. In [Birbach and Frese, 2009] an Unscented Kalman Filter, exploiting both position and size of the ball, was used to track balls for a humanoid robot. Finding correspondences between images was done using a multiple-hypothesis tracker [Reid, 1979].

6.2 Problem Formulation

The goal is to track an object with a known process model. This is to be done with images captured from different view points and where possibly no images are captured simultaneously. The object to be tracked is described by the discrete-time state-space model

$$x(k+1) = \Phi(k)x(k) + \Gamma(k)u(k) + v(k) \tag{6.1}$$

where x is the state vector and u is a known input signal. Measurements are assumed to be on the form

$$y(k) = C(k)x(k) + e(k) \tag{6.2}$$

Note that the C-matrix is time varying and will depend on the camera parameters and the image coordinates at that particular time step. The disturbances v and e are discrete-time white-noise processes with zero mean values and covariances

$$\mathrm{E}\left(\begin{bmatrix} v(i) \\ e(j) \end{bmatrix} \begin{bmatrix} v(i) \\ e(j) \end{bmatrix}^T \right) = \begin{bmatrix} R_v & R_{ve} \\ R_{ve}^T & R_e \end{bmatrix} \delta_{ij} \tag{6.3}$$

6.3 Method

State Estimation

A Kalman filter is used to estimate the state of the tracked object. The update law can be described by (6.4)–(6.9). Here, for example, $\hat{x}(k+1|k)$ denotes the estimate of x at sample $k+1$ based on measurements up to sample k, and $P(k+1|k)$ the covariance of that estimate.

$$\hat{x}(k|k) = \hat{x}(k|k-1) + K_f\left(y(k) - C(k)\hat{x}(k|k-1)\right) \tag{6.4}$$

$$K_f(k) = P(k|k-1)C^T(k)\left(R_e(k) + C(k)P(k|k-1)C^T(k)\right)^{-1} \tag{6.5}$$

$$P(k|k) = P(k|k-1) - K_f(k)C(k)P(k|k-1) \tag{6.6}$$

$$\begin{aligned} \hat{x}(k+1|k) = {} & \Phi(k)\hat{x}(k|k-1) + \Gamma(k)u(k) \\ & + K(k)\left(y(k) - C(k)\hat{x}(k|k-1)\right) \end{aligned} \tag{6.7}$$

$$\begin{aligned} K(k) = {} & \left(\Phi(k)P(k|k-1)C^T(k) + R_{ve}(k)\right) \\ & \cdot \left(R_e(k) + C(k)P(k|k-1)C^T(k)\right)^{-1} \end{aligned} \tag{6.8}$$

$$\begin{aligned} P(k+1|k) = {} & \Phi(k)P(k|k-1)\Phi^T(k) + R_v(k) \\ & - K(k)\left(\Phi(k)P(k|k-1)C^T(k) + R_{ve}(k)\right)^T \end{aligned} \tag{6.9}$$

The Measurement Function

From the position of a feature point in a single image it is possible to determine a ray in 3D-space along which the point must be situated, cf. Fig. 6.1. The ray parameters then have to be related to the process model states in some way in order to be used as input to the Kalman filter.

To determine the ray for a point in an image, first two lines through the point in the image are chosen. The easiest way to do this is to take one horizontal and one vertical line. Each one of these lines uniquely defines a plane through the line and the focal point. Finally, the ray can be determined as the intersection of these two planes.

Let $\mathbf{x}$ be an image point, and $\mathbf{X}$ a point in 3D space, both represented as homogeneous coordinate vectors. For an introduction to homogeneous coordinates, see Sec. A.1.

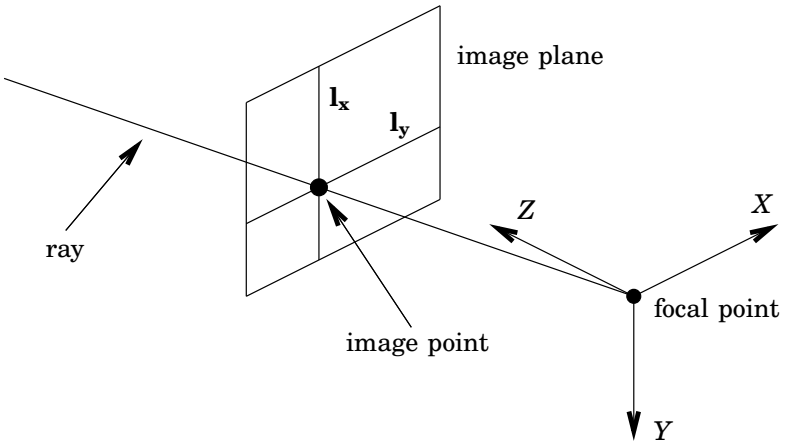


Figure 6.1 Illustration of a ray. An object projected onto some image point can be located anywhere along the corresponding ray. The coordinate system in this example image corresponds to the camera projection matrix $P_c = [I_3 \;\; 0_{3\times1}]$.

A line $\mathbf{l}$ in an image can be represented as the points fulfilling the equation

$$\mathbf{l}^T\mathbf{x} = 0 \tag{6.10}$$

where $\mathbf{l} = [l_1 \; l_2 \; l_3]^T$ and $\mathbf{x} = [x \; y \; 1]^T$. Similarly, a plane $\boldsymbol{\pi}$ in space can be represented as the points fulfilling

$$\boldsymbol{\pi}^T\mathbf{X} = 0 \tag{6.11}$$

where $\boldsymbol{\pi} = [\pi_1 \; \pi_2 \; \pi_3 \; \pi_4]^T$ and $\mathbf{X} = [X \; Y \; Z \; 1]^T$. If the point $\mathbf{X}$ projects onto the image point $\mathbf{x}$, then

$$\mathbf{x} \sim P_c\mathbf{X} \tag{6.12}$$

where P_c is the camera projection matrix. Inserting (6.12) into (6.10), it can be seen that all points projecting onto the line $\mathbf{l}$ satisfy

$$0 = \mathbf{l}^T\mathbf{x} \sim \mathbf{l}^T P_c\mathbf{X} = (P_c^T\mathbf{l})^T\mathbf{X} \tag{6.13}$$

i.e., all points projecting onto $\mathbf{l}$ reside in the plane defined by

$$\boldsymbol{\pi} = P_c^T\mathbf{l} \tag{6.14}$$

If the coordinate of a feature point in an image is (x, y), a convenient choice of lines through this point is $\mathbf{l_x} = [-1 \; 0 \; x]^T$ and $\mathbf{l_y} = [0 \; -1 \; y]^T$ (cf. Fig. 6.1) corresponding to the planes defined by

$$\begin{aligned} \boldsymbol{\pi_x} &= P_c^T\mathbf{l_x} \\ \boldsymbol{\pi_y} &= P_c^T\mathbf{l_y} \end{aligned} \tag{6.15}$$

The ray can then be described as the intersection of the planes $\boldsymbol{\pi_x}$ and $\boldsymbol{\pi_y}$.

The planes in (6.15) each puts the constraint $\boldsymbol{\pi}^T\mathbf{X} = 0$ on the feature point position $\mathbf{X}$. This can be rewritten as

$$0 = \boldsymbol{\pi}^T\mathbf{X} = \begin{bmatrix} \pi_1 & \pi_2 & \pi_3 & \pi_4 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \Longleftrightarrow$$

$$-\pi_4 = \begin{bmatrix} \pi_1 & \pi_2 & \pi_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \Longleftrightarrow$$

$$\begin{aligned} -\pi_4 &= cx' \\ \text{where } c &= \begin{bmatrix} \pi_1 & \pi_2 & \pi_3 \end{bmatrix} \\ \text{and } x' &= \begin{bmatrix} X & Y & Z \end{bmatrix}^T \end{aligned} \tag{6.16}$$

If $\boldsymbol{\pi}$ is normalized so that

$$\sqrt{\pi_1^2 + \pi_2^2 + \pi_3^2} = 1 \tag{6.17}$$

then $-\pi_4$ can be interpreted as the signed length of the orthogonal projection of the feature point position onto the direction $[\pi_1\ \pi_2\ \pi_3]$.

Each image with a measurement of a feature position gives two constraints, each specified by a row vector c in (6.16); one in the x-direction and one in the y-direction of the image. If several images are available, this can be handled simply by adding rows in the C-matrix of (6.2):

$$C = \begin{bmatrix} c_{ax} \\ c_{ay} \\ c_{bx} \\ c_{by} \\ \vdots \end{bmatrix} \tag{6.18}$$

where a and b denote different images and x and y the different measurement directions. Similarly, the measurement vector y of (6.2) is composed of the negative fourth component of the planes $\boldsymbol{\pi}$ in (6.15):

$$y = \begin{bmatrix} -\pi_{4ax} \\ -\pi_{4ay} \\ -\pi_{4bx} \\ -\pi_{4by} \\ \vdots \end{bmatrix} \tag{6.19}$$

If no measurements at all are available at some time step, the last term of Eqs. (6.4), (6.6), (6.7), and (6.9) simply disappears.

The algorithm for generating the measurement function for the Kalman filter can be summarized by the following steps:

1. For each image point, calculate the planes $\boldsymbol{\pi_x}$ and $\boldsymbol{\pi_y}$ according to Eq. (6.15).

2. Normalize the $\boldsymbol{\pi}$-vectors according to Eq. (6.17).

3. Stack the measurements to form C and y according to Eqs. (6.18) and (6.19).

6.4 Detection of False Positives

When using computer vision, two different kinds of noise are common. If the correct object was detected in the image, the value of the position estimate has a distribution with most of the probability mass close to the true value. There is also a risk that the position estimate was based on the wrong feature, which typically results in outliers, i.e., estimates with values far away from the main body of the distribution. For good tracker performance, the outliers should be identified and eliminated.

Outlier Detection

In this subsection the time index is left out in the notation, since it is the same for all quantities.

The measurement, y, is according to (6.2) given on the form

$$y = Cx + e \tag{6.20}$$

with $\mathrm{E}[e] = 0$ and $\mathrm{E}[ee^T] = R_e$. Based on the state estimate, $\hat{x}$, the expected value of the measurement is

$$\hat{y} = C\hat{x} \tag{6.21}$$

The state-estimate error is given by $\tilde{x} = x - \hat{x}$, with $\mathrm{E}[\tilde{x}] = 0$, $\mathrm{E}[\tilde{x}\tilde{x}^T] = R_x$, and $\mathrm{E}[\tilde{x}e^T] = 0$. The measurement error can be estimated by

$$\begin{aligned} \tilde{y} &= y - \hat{y} && (6.22)\\ &= Cx + e - C\hat{x} && (6.23)\\ &= C\tilde{x} + e && (6.24) \end{aligned}$$

with the mean value $\mathrm{E}[\tilde{y}] = 0$ and the covariance

$$\begin{aligned} R_y &= \mathrm{E}[\tilde{y}\tilde{y}^T] \\ &= \mathrm{E}\left[(C\tilde{x} + e)\left(\tilde{x}^T C^T + e^T\right)\right] \\ &= CR_xC^T + R_e \end{aligned} \tag{6.25}$$

where the first term originates from the uncertainty of the state estimate and the second term originates from the measurement uncertainty.

Using Gaussian approximations for all distributions, the probability density function of $\tilde{y}$ is

$$f(\tilde{y}) = \frac{1}{(2\pi)^{n/2}\left|R_y\right|^{1/2}} \exp\left(-\frac{1}{2}\tilde{y}^T R_y^{-1}\tilde{y}\right) \tag{6.26}$$

where n is the dimension of the state vector. For constant R_y, (6.26) is a decreasing function of $\tilde{y}^T R_y^{-1}\tilde{y}$. Motivated by this fact, it is a natural option to consider the measurement as an outlier if a condition on the form

$$\tilde{y}^T R_y^{-1}\tilde{y} > p^2 \tag{6.27}$$

is fulfilled, where p is a tuning parameter. Equation (6.27) defines an ellipsoid. Inserting (6.27) into (6.26) it can be seen that all points outside the decision boundary have a lower probability density than all points inside the boundary.

Managing of Multiple Trajectory Hypotheses Simultaneously

It is useful to have a layer on top of the Kalman filter, keeping track of several state vectors, each representing the trajectory of one ball. The obvious advantage of this is that the trajectories of several balls can be tracked simultaneously. A more important advantage has to do with outlier detection. In each iteration of the Kalman filter, the measurements, y, are discarded if they are not close to the expected measurement, $\hat{y}$, based on the state estimates, as described in the previous subsection. If the measurement initiating the trajectory is a false positive, successive correct measurements will be discarded, since they are not close to what is expected after the incorrect measurement. This means that one incorrect measurement will block the system. The algorithm used to handle this situation is described by the following pseudo code, which is run on

every measurement:

```
for all trajectories
    if the measurement is not an outlier to this trajectory
        perform measurement update of Kalman filter
    if the trajectory has not received a measurement for a while
        throw it away
if the measurement did not match any existing trajectory
    create a new trajectory
```

6.5 Results

Figure 6.2 shows example images of a thrown ball from the perspectives of two cameras. The detected positions of the ball from a sequence of such images are marked with green dots connected by lines. Different subsets of these data were used as input to the tracker described in this chapter. The results are shown in Figs. 6.3–6.6. For all cases the estimate was initialized by the method described in Chapter 5. The model (2.5) with the state vector $x = [X \;\; Y \;\; Z \;\; \dot{X} \;\; \dot{Y} \;\; \dot{Z}]^T$ was used. The vector x was equal to x' in Eq. (6.16) extended by three velocity states. Hence, the C-matrix of Eq. (6.18) could be used with x by extending it with three columns of zeros to the right. The sampling time used was $h = 20$ ms and

$$R_v = 0.001^2 \cdot \text{diag}[\; 2^2 \quad 2^2 \quad 2^2 \quad 5^2 \quad 5^2 \quad 5^2 \;]\ \text{m}^2 \tag{6.28}$$

$$R_{ve} = 0 \tag{6.29}$$

$$R_e = \text{diag}[\; 0.01^2 \quad 0.01^2 \;]\ \text{m}^2 \tag{6.30}$$

The accuracy of the estimates are illustrated by ellipsoids on the form $(z - \hat{z})^T R^{-1} (z - \hat{z}) = p^2$, where $z \in \mathbb{R}^{3\times 1}$ is the position or velocity and R is the covariance matrix of its estimate, $\hat{z}$. In Figs. 6.3–6.6, the value of the confidence level parameter in (6.27) was $p = 3$, which means that there is a 97 % probability that the real value was within the ellipsoid, assuming that the distributions were Gaussian.

Figures 6.3 and 6.4 show the estimates based on all the data points in Fig. 6.2. The position estimates formed a smooth parabola and the estimated velocities decreased in the vertical direction linearly with time, as expected by the model due to the gravity. The size of the uncertainty ellipsoids decreased as the number of measurements increased. The tracker classified one measurement as a false positive, which is indicated by a red circle in Fig. 6.2.

It was possible to track the ball using only a single fixed camera, since the ball was observed from slightly different angles as it moved across the

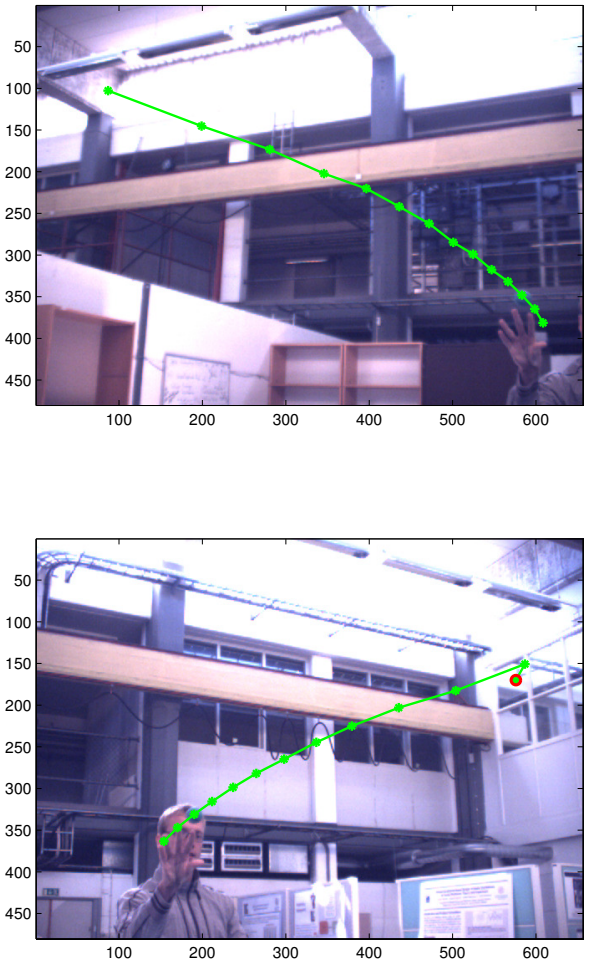


Figure 6.2 Example images from a sequence of images of a thrown ball, as seen from the two different cameras. The detected ball positions from the entire sequence are marked with green dots. A detected ball position that was classified as an outlier by (6.27) is marked with a red circle. In these example images the ball can be seen at the third green dot in the trajectory.

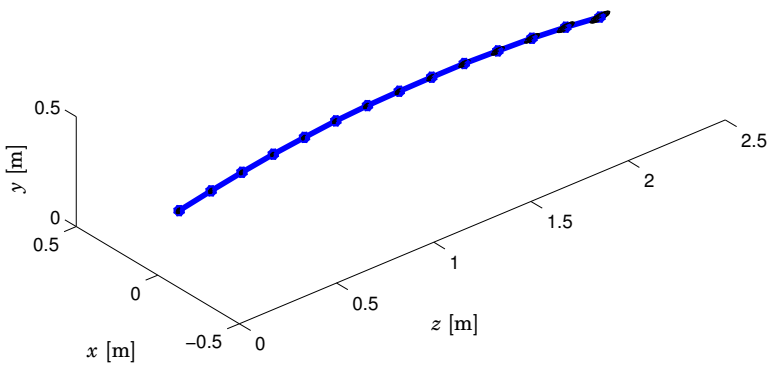


Figure 6.3 Position estimates based on all measured image coordinates from both cameras in Fig. 6.2.

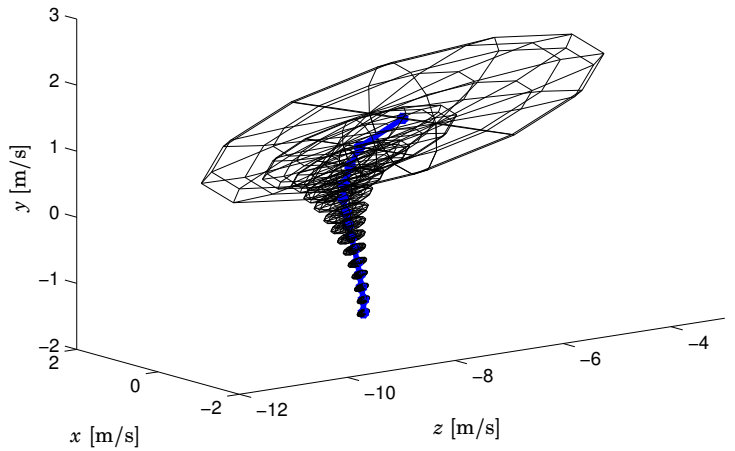


Figure 6.4 Velocity estimates based on all measured image coordinates from both cameras in Fig. 6.2.

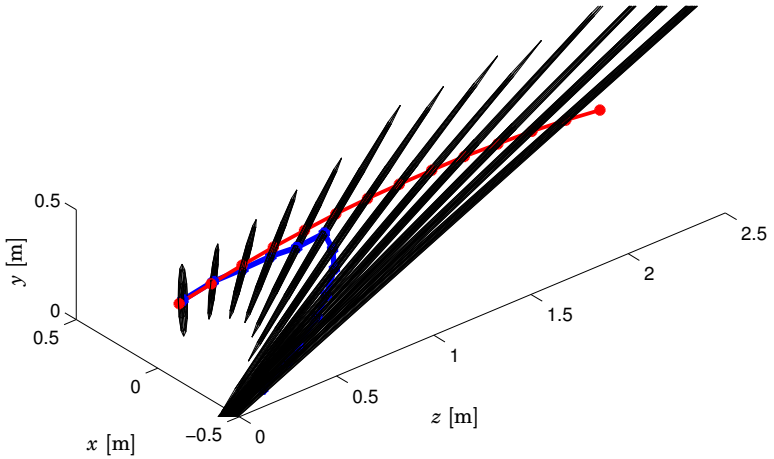


Figure 6.5 The blue curve shows position estimates based on the measured image coordinates from only the upper image in Fig. 6.2. For comparison, estimates based on the data from both images of Fig. 6.2 are shown in red.

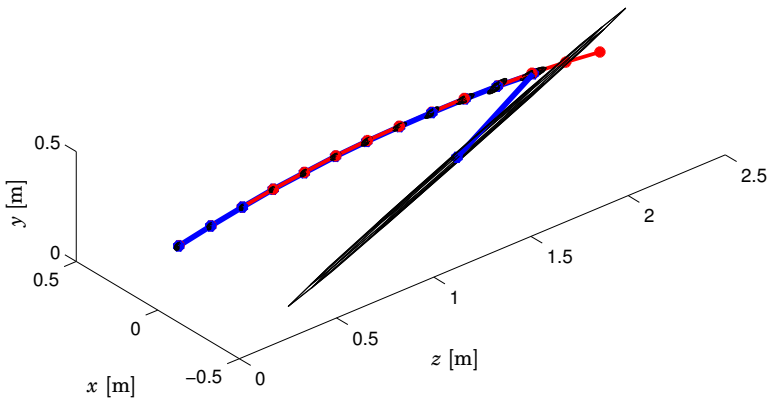


Figure 6.6 The blue curve shows estimates based on data from only one camera for odd time indices, and data from another camera for even time indices. Estimates based on all the data from both cameras are shown in red.

image. In Fig. 6.5 the blue curve shows the estimates based only on measurements from the lower image in Fig. 6.2. The uncertainty ellipsoids were initially very oblong, since the position in the direction of the rays was very uncertain. As the ball moved and got observed from different angles, the estimate variance decreased drastically. The estimated trajectory based on the images from both cameras is plotted in red for reference. The uncertainty ellipsoids of the estimates in the blue trajectory enclosed the corresponding estimates in the red trajectory as expected. When both cameras were used, a full state estimate could be obtained after two time samples, while three time samples were needed when only one camera was used. Hence, the first point in the red trajectory had no corresponding position estimate in the blue trajectory.

When the blue trajectory in Fig. 6.6 was generated, only the data from the odd time indices in the upper image and the data from the even time indices in the lower image of Fig. 6.2 were used. After four measurements, the estimate converged to almost coincide with the red curve, generated by all the data points in both images of Fig. 6.2, and the 97 % confidence ellipsoids are almost too small to be visible. The example illustrates that the filter works well if images from cameras at two different locations are provided, even if the images are not captured simultaneously.

The tracker was used to track balls for the ball-catching robot described in Chapter 8.

6.6 Discussion

No explicit triangulation was done in the proposed procedure. However, if the rows in C of (6.18) corresponded to measurements in many different directions over time, an estimate with low variance in all directions could be obtained.

A benefit of the tracking approach described in this chapter is that it easily handles any number of cameras. In (6.18) and (6.19) two more rows are simply added to C and y for every camera that captured an image at the same sampling instant. Another advantage is that a measurement can be used even if there are no valid measurements from any other cameras, so that no 3D position estimate can be made from the data from only that sampling instant. It was shown that a ball could be tracked using a single static camera. Accuracy was, however, significantly better if two cameras at different positions were used. The accuracy could probably be improved further by using even more cameras.

The position of a 3D point and its image projection are related by a non-linear measurement function. The standard way to handle the non-linearity in an Extended Kalman Filter [Schmidt, 1966] is to linearize the

measurement around the current state estimate. The method presented in this chapter uses a linearization that depends only on the image coordinate and is independent of the state estimate.

If a continuous-time model of the process is available, the procedure can be generalized to completely aperiodic measurements. The discrete-time model representation (6.1)–(6.3) then has to be recalculated at every measurement to reflect the actual time that passed since the previous measurement.

The described managing of multiple trajectories works well for handling false positive ball detections. For example, a trajectory hypothesis induced by false positives can be kept in the memory until it can be confidently discarded, and in the meantime a trajectory based on true measurements can be estimated. The process of initiating new trajectory hypotheses does, however, not work well if several balls are detected in each image. Though several balls can be tracked, the performance will not be good unless the sequence of which balls are detected in which images follows an advantageous pattern. Extending the system to handle several balls in a better way is part of the future work. Possible ways to do this could be to use a multiple-hypotheses tracker (MHT) [Birbach and Frese, 2009] or a probability-hypothesis-density (PHD) filter [Birbach and Frese, 2011].

6.7 Conclusions

This chapter describes a strategy for tracking dynamical objects with computer vision. A way of mapping image space data to a Kalman filter in 3D space is described. It provides a simple way to use data from an arbitrary number of pictures captured simultaneously. In combination with a dynamical model of the tracked object, it enables determination of the position of the object in 3D space without any two images having to be captured simultaneously. It also allows tracking of objects in 3D space, using only a single static camera. Furthermore, this chapter presents a method for determining whether a measurement is an outlier, based on the covariances of the measurement and the state estimate.

7

Robot Trajectory Generation with Uncertain Target Point

7.1 Introduction

In an unpredictable environment a robot may need to react quickly to events and measurements, and generate a new trajectory for its motion. One such application is a ball-catching robot, presented in Chapter 2. The tracker described in Chapter 6 can be used to estimate the position where a ball should be caught. When few measurements are available, the variance of the estimated state of the ball may be large, but as the ball moves along the trajectory and more measurements of the ball are collected, the accuracy of the estimated catching point increases. In order to have good performance of the ball-catcher, the robot should start moving the box toward the catching pose as soon as there is an estimate, and then modify the trajectory on the fly when better estimates of the target pose become available. Care should also be taken to make the trajectories smooth to avoid excessive wear on the mechanical parts. The method presented in this chapter does so by minimizing the maximum acceleration during the motion, but allows the jerk (the derivative of the acceleration) to be infinite, resulting in piecewise second-order trajectories. It will be discussed why a trajectory with infinite jerk can be considered smooth. The computation time for the resulting trajectory generation is only a few microseconds.

Popular methods for robotic path planning include probabilistic roadmaps [Geraerts and Overmars, 2002] and rapidly exploring random trees [Lavalle and Kuffner, 2000]. They solve the problem of finding a path past obstacles, but when there are no obstacles within the robot's work space this is an unnecessary and time-consuming step. Off-line methods can be used to optimize trajectories for sophisticated optimization criteria [Debrouwere et al., 2013]. In [Dahl and Nielsen, 1989], the

path speed was computed on line taking torque limitations into account, but only for pre-defined paths.

The work described in this chapter is in line with the philosophy presented in [Kröger et al., 2006], which suggests to use an MPC-like strategy for reactive robot trajectory generation. MPC (Model Predictive Control) [Maciejowski, 2002] solves an optimal-control problem in each control cycle and uses the first part of the solution. In the next control cycle, the remainder of the solution is discarded, and a new solution is calculated based on new measurement data. The standard method is to use sampled systems and optimize an objective function numerically to calculate a discrete-time control output for a given time horizon. Though this allows to solve a wide range of problems, the general method may not be suitable for real-time high-resolution trajectory generation in robotics, where the solution should be available within milliseconds. For this application, specialized fast optimization methods are needed.

The work presented in [Kröger et al., 2006; Haschke et al., 2008] used approaches similar to the one presented in this chapter. They also generated piecewise second-order trajectories with limitations on speed and acceleration, but they did so from the perspective of time-optimal trajectories, i.e., they calculated the fastest possible trajectory not violating the constraints on velocity and acceleration, but they had no concern for minimizing the acceleration. Since they added the requirement that all joints of the robot should reach their targets at the same time, they also solved the non-time-optimal case, i.e., the case where a joint should reach its target at a specified time after the earliest possible time. When doing so, however, they used the maximum acceleration and reduced the maximum velocity. A further development was presented in [Kröger and Wahl, 2010], which allowed the jerk to be limited. For the non-time-optimal case, however, it used the maximum jerk and acceleration, and reduced the maximum speed. A near-optimal planner handling bounded jerks was presented already in [Macfarlane, 2001], but only for time-optimal trajectories.

An alternative for fast trajectory generation is to use third-order polynomials and specify the position and velocity at the beginning and the end of the trajectory, as done in [Hong, 1995]. Smoother trajectories can be achieved by using fifth-order polynomials and specifying also the acceleration at the beginning and end of the trajectory, as done in [Bätz et al., 2010; Lippiello and Ruggiero, 2012b]. One drawback with this approach is that it does not consider limitations on velocity and acceleration.

In [Bäuml et al., 2010] it was implied that they solved a problem identical to what is described in this chapter, though no details on the solution were given. Whereas my detailed solution to the problem was first published in [Linderoth, 2011], it was demonstrated to work already in

[Linderoth, 2009]. In [Bäuml et al., 2010] they proceeded to solve a more complex problem, optimizing the catching point along the trajectory of the ball, as well as redundancy resolution of the robot. It resulted in a computationally demanding solution running on 32 CPU cores, utilizing a pipeline structure to calculate new trajectories every 20 ms with latencies up to 60 ms. The high latencies resulted in shorter times to perform the motion and also reduced the catch rate, since the accuracy of the estimated catch pose typically increased over time.

The trajectories generated by the method described in this chapter have piecewise constant acceleration and unbounded jerk. One drawback with not bounding the jerk is that it may excite mechanical resonances of the robot, thus reducing the actual tracking performance. It will, however, be shown that infinite-jerk trajectories can compete well with bounded-jerk trajectories in terms of tracking performance. In some cases they even perform better than bounded-jerk trajectories.

7.2 Problem Formulation

The task is to generate a trajectory for moving a robot from one pose to another and reach the final position before a given deadline. While the robot is moving along the trajectory, the final point may change both in space and time and the trajectory should be modified accordingly.

In order to reduce wear on the mechanical parts it is desirable not to shake the robot excessively, even if the target point is updated frequently.

7.3 Method

The trajectory generation is performed separately for each DOF (degree of freedom). In the simplest case, each DOF corresponds to a robot joint, and in this case the generated trajectories can be used as references to the respective joints directly. If the trajectory generation is done in Cartesian space, the trajectories can be transformed to joint space by means of the inverse kinematics and the robot Jacobian.

The trajectory generation is performed for every DOF every time new values of the destination point and the deadline become available. If one DOF can not meet the deadline, the other DOFs are slowed down so they all reach their end points at the same time.

Assumptions

When a new destination point is acquired at time t_0 the joint has an initial position x_0 and initial velocity v_0. The time left to the deadline, t_d, is

$$T_d = t_d - t_0 \tag{7.1}$$

and the final (destination) position is x_f, where the velocity should be zero. The velocity and acceleration are limited to V and A respectively. The total distance left to go at time t_0 (when the new destination position is acquired) is denoted by

$$X = x_f - x_0. \tag{7.2}$$

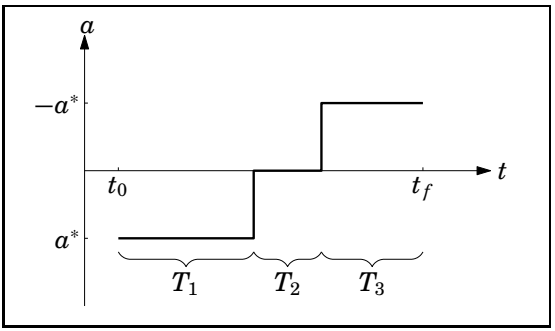


Figure 7.1 Illustration of acceleration profile

Trajectory Primitive

In the following, two strategies for trajectory generation will be described. They both split the movement into three time intervals:

1. apply acceleration a^* during time T_1
2. apply acceleration 0 during time T_2
3. apply acceleration $-a^*$ during time T_3

The acceleration profile is illustrated in Fig. 7.1. This trajectory primitive can be seen as bang-bang control in the acceleration, used to minimize the maximum acceleration when moving a given distance within a given time. Graphical examples of the resulting velocity profiles are given in Figs. 7.2–7.5 on the following pages. The middle time interval with zero acceleration is needed in case the velocity reaches its maximum value.

Let v_t be the velocity in the middle time interval, when the acceleration is zero. Since the acceleration has different signs before and after this interval, v_t is always an extremum of the velocity, and is hence convenient for determining whether an attempt to generate a trajectory violates the velocity constraints.

Strategy 1: Minimize Maximum Acceleration

This subsection describes a strategy that uses the smallest possible acceleration needed to reach the destination before the deadline. If the deadline cannot be met, the destination will be reached at time t_f, with $t_f > t_d$. The smallest possible t_f that does not violate the constraints on velocity and acceleration will then be used.

The problem can be formally described by equations (7.3)–(7.13), where $x(t)$, $v(t)$ and $a(t)$ are the position, velocity and acceleration as functions of time.

- Initial conditions:

$$x(t_0) = x_0 \tag{7.3}$$
$$v(t_0) = v_0 \tag{7.4}$$

- Dynamics:

$$v(t) = v_0 + \int_{t_0}^{t} a(t)\,dt \tag{7.5}$$
$$x(t) = x_0 + \int_{t_0}^{t} v(t)\,dt \tag{7.6}$$

- Constraints:

$$x(t_f) = x_f \tag{7.7}$$
$$v(t_f) = 0 \tag{7.8}$$

$$T_1 + T_2 + T_3 = t_f - t_0 \tag{7.9}$$
$$T_1 \geq 0, \quad T_2 \geq 0, \quad T_3 \geq 0 \tag{7.10}$$

$$-V \leq |v(t)| \leq V, \forall t \tag{7.11}$$
$$-A \leq |a(t)| \leq A, \forall t \tag{7.12}$$

- Parameterization of solution (cf. Fig. 7.1):

$$a(t) = \begin{cases} a^*, & t_0 & \leq t < t_0 + T_1 \\ 0\ , & t_0 + T_1 & \leq t < t_0 + T_1 + T_2 \\ -a^*, & t_0 + T_1 + T_2 & \leq t \leq t_0 + T_1 + T_2 + T_3 = t_f \end{cases} \tag{7.13}$$

Task: The goal is to find T_1, T_2, T_3 and a^* that solve (7.3)–(7.13). The quantities x_0, x_f, v_0, t_0, t_f, V, and A are given as a part of the problem formulation. Note that a^* can be either positive or negative.

Inserting the acceleration profile (7.13) into the dynamics equation (7.5) and enforcing the velocity end constraint (7.8) gives

$$v(t_f) = v_0 + \int_{t_0}^{t_f} a(t)\,dt \iff \tag{7.14}$$

$$0 = v_0 + a^*(T_1 - T_3) \tag{7.15}$$

Inserting the acceleration profile (7.13) into the dynamics equation (7.6), using the definition (7.2), and enforcing the position end constraint (7.7) gives

$$x(t_f) - x_0 = \int_{t_0}^{t_f} v(t)\,dt \iff \tag{7.16}$$

$$X = v_0(T_1 + T_2 + T_3) + a^*\left(\frac{T_1^2}{2} + T_1T_2 + T_1T_3 - \frac{T_3^2}{2}\right) \tag{7.17}$$

Equations (7.15) and (7.17) will be the foundation for calculating the trajectories.

For convenience we also derive a few equations including v_t, the extremum velocity attained during the zero-acceleration phase, cf. Fig. 7.1. The velocity v_t can be calculated by using the beginning of the acceleration profile and the dynamics equation (7.5):

$$v_t = v_0 + \int_{t_0}^{t} a(t)\,dt\Big|_{t_0+T_1 \le t < t_0+T_1+T_2} \iff \tag{7.18}$$

$$v_t = v_0 + a^*T_1 \iff \tag{7.19}$$

$$T_1 = (v_t - v_0)/a^* \tag{7.20}$$

Subtracting (7.15) from (7.19) gives

$$v_t = a^*T_3 \iff \tag{7.21}$$

$$T_3 = v_t/a^* \tag{7.22}$$

The solution of the trajectory generation will be divided into four cases, depending on whether the velocity and acceleration become saturated. The velocity profiles for the different cases are illustrated in Figs. 7.2–7.5. The total distance to go, X, is equal to the area of the shaded region above the t-axis, subtracted by the area of the shaded region below the t-axis. For each case a^*, v_t, and T_2 will be calculated. T_1 and T_3 can then be obtained from (7.20) and (7.22).

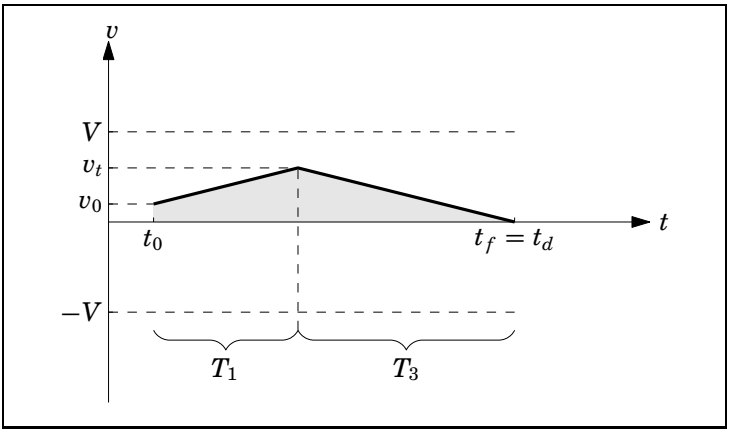


Figure 7.2 Illustration of velocity profile for Case 1.

Case 1: No saturation of velocity, no saturation of acceleration. In this case it is assumed that the deadline is met. It is also assumed that the velocity will not be saturated, thus requiring no phase with maximum speed and zero acceleration. The assumptions can be expressed as

$$t_f = t_d \tag{7.23}$$

$$T_2 = 0 \tag{7.24}$$

and these constraints will be enforced for Eqs. (7.25)–(7.33). For an example velocity profile, see Fig. 7.2.

Combining (7.23) and (7.24) with (7.9) and (7.1) gives

$$T_d = T_1 + T_3 \tag{7.25}$$

Inserting the expressions (7.20), (7.24), and (7.22) for T_1, T_2, and T_3, into (7.17) and (7.25) gives

$$X = \frac{2v_t^2 - v_0^2}{2a^*} \tag{7.26}$$

$$T_d = \frac{2v_t - v_0}{a^*} \tag{7.27}$$

We now have two equations–(7.26) and (7.27)–with two unknowns–a^* and

v_t. By taking X/T_d, we can eliminate a^*, resulting in

$$\begin{aligned}
\frac{X}{T_d} &= \frac{v_t^2 - v_0^2/2}{2v_t - v_0} \iff \\
0 &= v_t^2 - 2\frac{X}{T_d}v_t + \frac{X}{T_d}v_0 - \frac{v_0^2}{2} \\
&= \left(v_t - \frac{X}{T_d}\right)^2 - \frac{X^2}{T_d^2} + \frac{X}{T_d}v_0 - \frac{v_0^2}{2} \\
&= \left(v_t - \frac{X}{T_d}\right)^2 - \left(\frac{X}{T_d} - \frac{v_0}{2}\right)^2 - \left(\frac{v_0}{2}\right)^2 \iff \\
v_t &= \frac{X}{T_d} + k\sqrt{\left(\frac{X}{T_d} - \frac{v_0}{2}\right)^2 + \left(\frac{v_0}{2}\right)^2}
\end{aligned} \tag{7.28}$$

where $k = \pm 1$. To determine the sign of k, consider (7.20) and (7.22). In order for the times to be non-negative, as required by (7.10), both v_t and $(v_t - v_0)$ must have the same sign as a^* or be zero, which implies

$$0 \leq v_t(v_t - v_0) \tag{7.29}$$

Inserting (7.28) into (7.29) gives after some algebra

$$0 \leq 2\left(\frac{X}{T_d} - \frac{v_0}{2}\right)\left(\left(\frac{X}{T_d} - \frac{v_0}{2}\right) + k\sqrt{\left(\frac{X}{T_d} - \frac{v_0}{2}\right)^2 + \left(\frac{v_0}{2}\right)^2}\right) \tag{7.30}$$

Since

$$\sqrt{\left(\frac{X}{T_d} - \frac{v_0}{2}\right)^2 + \left(\frac{v_0}{2}\right)^2} \geq \left|\frac{X}{T_d} - \frac{v_0}{2}\right| \tag{7.31}$$

(7.30) is fulfilled for

$$k = \begin{cases} 1, & X/T_d \geq v_0/2 \\ -1, & X/T_d \leq v_0/2 \end{cases} \tag{7.32}$$

If $X/T_d = v_0/2$, then either sign of k is valid. The difference it will make is whether $T_1 = 0$ and $v_t = v_0$ or alternatively $T_3 = 0$ and $v_t = 0$. In both cases $a(t)$ is the same.

Inserting (7.32) into (7.28) lets us calculate v_t. If $|v_t| > V$, no valid solution for Case 1 exists.

Solving Eq. (7.27) for the acceleration gives

$$a^* = \frac{2v_t - v_0}{T_d} \tag{7.33}$$

If $|a^*| > A$, no valid solution for Case 1 exists.

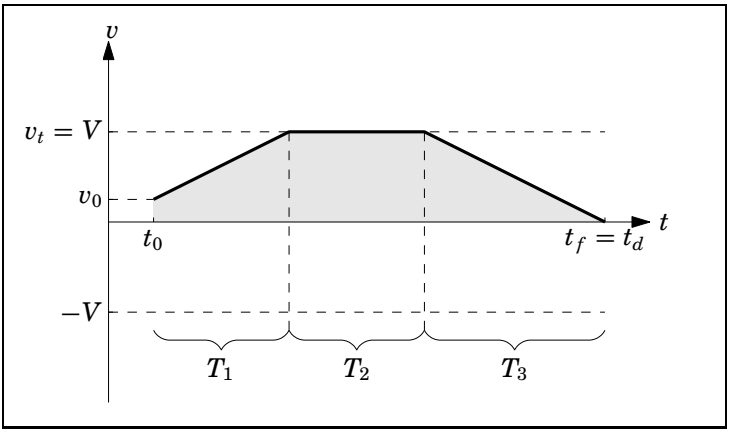


Figure 7.3 Illustration of velocity profile for Case 2; with saturated velocity.

Case 2: Saturation of velocity, no saturation of acceleration. In this case it is assumed that the deadline is met. It also assumes that the velocity will be saturated, so there is a part with maximum velocity in the trajectory. The assumptions can be expressed as

$$t_f = t_d \iff T_d = T_1 + T_2 + T_3 \tag{7.34}$$

$$v_t = V \operatorname{sign}(X) \tag{7.35}$$

and these constraints will be enforced for Eqs. (7.36)–(7.38). For an example velocity profile, see Fig. 7.3

Extracting T_2 from (7.34) and inserting it into (7.17) gives

$$\begin{aligned} X &= v_0 T_d + a^* \left(\frac{T_1^2}{2} + T_1(T_d - T_1 - T_3) + T_1 T_3 - \frac{T_3^2}{2} \right) \\ &= (v_0 + a^* T_1) T_d - a^* \left(\frac{T_1^2}{2} + \frac{T_3^2}{2} \right) \\ &= v_t T_d - \frac{(v_t - v_0)^2 + v_t^2}{2a^*} \end{aligned} \tag{7.36}$$

where the last equality follows from (7.20) and (7.22). Solving (7.36) for a^* gives

$$a^* = \frac{(v_t - v_0)^2 + v_t^2}{2(v_t T_d - X)} \tag{7.37}$$

Now we have v_t and a^* and can get T_1 and T_3 from (7.20) and (7.22), and get T_2 from (7.34). For the solution to be valid we must have $T_3 \geq 0$,

which in combination with (7.22) gives that v_t and a^* must have equal sign. Applied to (7.37) this gives that v_t and $(v_t T_d - X)$ must have equal sign, which in combination with (7.35) gives

$$|X| < VT_d \iff \frac{|X|}{T_d} < V \tag{7.38}$$

If (7.38) is not fulfilled, no valid solution to Case 2 exists. This requirement is very intuitive, since it means that the mean velocity must be less than the maximum allowed velocity. For a valid solution to exist, it is also required that $|a^*| \leq A$.

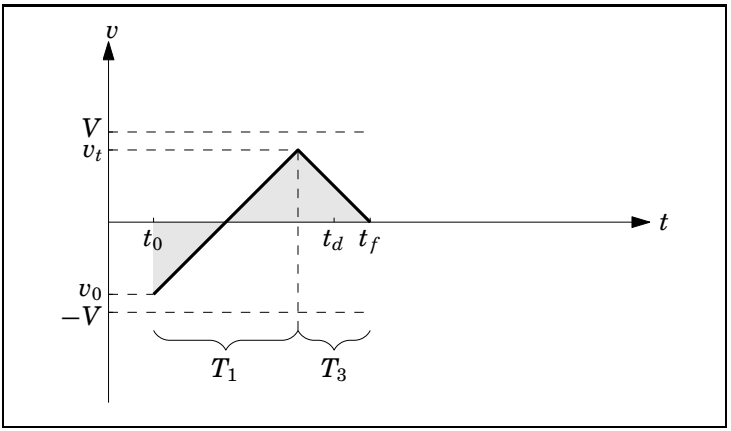


Figure 7.4 Illustration of velocity profile for Case 3; with saturated acceleration (maximum slope during T_1 and T_3) and missed deadline, t_d.

Case 3: No saturation of velocity, saturation of acceleration. In this case it is assumed that the velocity will not be saturated and that the maximum acceleration will be used. These assumptions can be expressed as

$$T_2 = 0 \tag{7.39}$$

$$a^* = \pm A \tag{7.40}$$

and these constraints will be enforced for Eqs. (7.41)–(7.53). The deadline will not be met. The sign of a^* will be determined later. For an example velocity profile, see Fig. 7.4

The conditions for Eq. (7.26) being valid are fulfilled, and solving for v_t gives

$$v_t = k\sqrt{Xa^* + v_0^2/2} \tag{7.41}$$

where $k = \pm 1$. It will now be shown that there is always exactly one solution to Eq. (7.41) fulfilling

$$Xa^* + v_0^2/2 \geq 0 \tag{7.42}$$
$$T_1 = (v_t - v_0)/a^* \geq 0 \tag{7.43}$$
$$T_3 = v_t/a^* \geq 0 \tag{7.44}$$

where Eq. (7.42) comes from the requirement that Eq. (7.41) should be real, and Eq. (7.43) and Eq. (7.44) come from the requirement that all durations must be non-negative. The condition (7.44) gives

$$\begin{aligned} &0 \leq \frac{v_t}{a^*} = \frac{k}{a^*}\sqrt{Xa^* + v_0^2/2} \iff \\ &k = \text{sign}(a^*) \end{aligned} \tag{7.45}$$

- Make the assumption $\boxed{a^* = A > 0}$.
  Through (7.41) and (7.45) this gives $v_t \geq 0$. Condition (7.42) is equivalent to

  $$2XA \geq -v_0^2 \tag{7.46}$$

  Condition (7.43) is trivial to verify if $v_0 \leq 0$, and in the case $v_0 > 0$ it is equivalent to

  $$\begin{aligned} v_t - v_0 \geq 0 &\iff \\ \sqrt{XA + v_0^2/2} \geq v_0 &\iff \\ XA + v_0^2/2 \geq v_0^2 &\iff \\ 2XA \geq v_0^2 \end{aligned} \tag{7.47}$$

  Condition (7.46) is valid for all v_0 and (7.47) only for $v_0 > 0$. Both conditions are fulfilled if and only if

  $$2XA \geq v_0|v_0| \tag{7.48}$$

- Make the assumption $\boxed{a^* = -A < 0}$.
  Through (7.41) and (7.45) this gives $v_t \leq 0$. Condition (7.42) is equivalent to

  $$2XA \leq v_0^2 \tag{7.49}$$

Condition (7.43) is trivial to verify if $v_0 \geq 0$ and in the case $v_0 < 0$ it is equivalent to

$$\begin{aligned} v_t - v_0 \leq 0 &\iff \\ -\sqrt{-XA + v_0^2/2} \leq v_0 &\iff \\ -XA + v_0^2/2 \geq v_0^2 &\iff \\ 2XA \leq -v_0^2 & \end{aligned} \tag{7.50}$$

Condition (7.49) is valid for all v_0 and (7.50) only for $v_0 < 0$. Both conditions are fulfilled if and only if

$$2XA \leq v_0|v_0| \tag{7.51}$$

Since either (7.48) or (7.51) is fulfilled for all values of X and v_0, there is always a solution for either $a^* = A$ or $a^* = -A$. On the curve $2XA = v_0|v_0|$ both values of a^* are valid. The sign of a^* will only affect whether $T_1 = 0$ or $T_3 = 0$ and in both cases $a(t)$ will be the same.

Inserting (7.41) into (7.44) gives

$$\begin{aligned} T_3 &= \frac{v_t}{a^*} \\ &= \frac{\text{sign}(a^*)}{a^*}\sqrt{Xa^* + v_0^2/2} \\ &= \frac{1}{A}\sqrt{Xa^* + v_0^2/2} \end{aligned} \tag{7.52}$$

Combining (7.43) and (7.44) gives

$$\begin{aligned} T_1 &= \frac{v_t}{a^*} - \frac{v_0}{a^*} \\ &= T_3 - \frac{v_0}{a^*} \end{aligned} \tag{7.53}$$

Equation (7.41) lets us calculate v_t. If $|v_t| > V$, no valid solution for Case 3 exists.

Case 4: Saturation of velocity, saturation of acceleration. In this case it is assumed that both velocity and acceleration become saturated, giving the constraints

$$v_t = V\,\text{sign}(X) \tag{7.54}$$

$$a^* = A\,\text{sign}(X) \tag{7.55}$$

and these assumptions will be enforced for (7.56)–(7.57). The deadline will not be met.

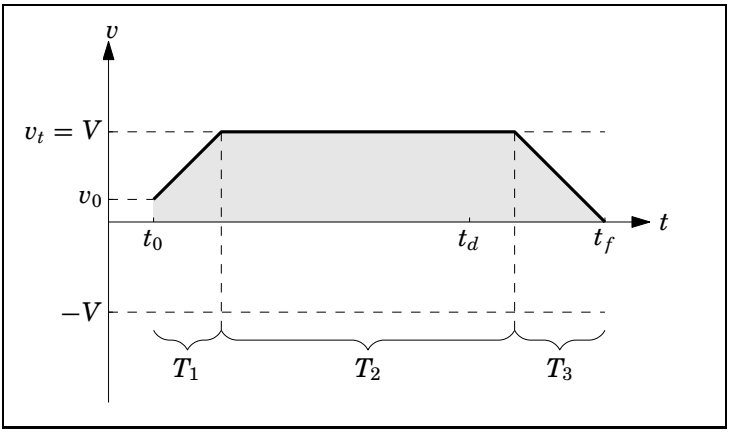


Figure 7.5 Illustration of velocity profile for Case 4; with saturated velocity and saturated acceleration (maximum slope during T_1 and T_3), and missed deadline, t_d.

Inserting (7.20) and (7.22) into (7.17) gives

$$\begin{aligned} X &= \frac{v_t v_0 - v_0^2}{a^*} + v_0 T_2 + \frac{v_t v_0}{a^*} \\ &\quad + a^* \left(\frac{v_t^2 - 2v_t v_0 + v_0^2}{2a^{*2}} + \frac{T_2}{a^*}(v_t - v_0) + \frac{v_t^2 - v_t v_0}{a^{*2}} - \frac{v_t^2}{2a^{*2}} \right) \\ &= v_t T_2 + \frac{2v_t^2 - v_0^2}{2a^*} \end{aligned} \tag{7.56}$$

Solving (7.56) for T_2 gives

$$\begin{aligned} T_2 &= \frac{X}{v_t} - \frac{v_t}{a^*} + \frac{v_0^2}{2v_t a^*} \\ &= \frac{|X|}{V} - \frac{V}{A} + \frac{v_0^2}{2VA} \end{aligned} \tag{7.57}$$

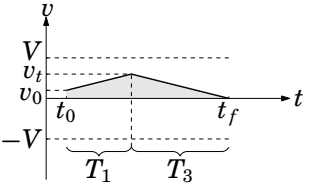
The durations T_1 and T_3 can be obtained from (7.20) and (7.22).

Summary of solution. In the following, the solution to Eqs. (7.3)–(7.13) for different cases of saturation will be summarized. First try using Case 1, where neither velocity nor acceleration are saturated.

- **Case 1: velocity not saturated, acceleration not saturated**

$$v_t = \begin{cases} X/T_d + \sqrt{(X/T_d - v_0/2)^2 + (v_0/2)^2}, & X/T_d \geq v_0/2 \\ X/T_d - \sqrt{(X/T_d - v_0/2)^2 + (v_0/2)^2}, & X/T_d \leq v_0/2 \end{cases}$$

$$a^* = (2v_t - v_0)/T_d$$
$$T_1 = (v_t - v_0)/a^*$$
$$T_2 = 0$$
$$T_3 = v_t/a^*$$

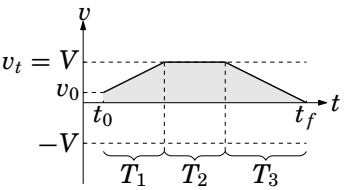


If $|v_t| > V$ the velocity constraint is violated and the solution is not valid. Try using Case 2.

If $|a^*| > A$ the acceleration constraint is violated and the solution is not valid. Try using Case 3. The deadline will not be met.

- **Case 2: velocity saturated, acceleration not saturated**

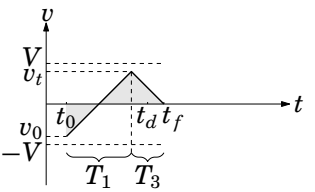
$$v_t = V\,\mathrm{sign}(X)$$
$$a^* = \frac{(v_t - v_0)^2 + v_t^2}{2(v_t T_d - X)}$$
$$T_1 = (v_t - v_0)/a^*$$
$$T_3 = v_t/a^*$$
$$T_2 = T_d - T_1 - T_3$$



If $VT_d < |X|$ or $|a^*| > A$ the solution is not valid. Use Case 4. The deadline will not be met.

- **Case 3: velocity not saturated, acceleration saturated**

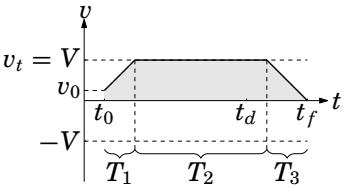
$$a^* = \begin{cases} A, & 2XA \geq v_0|v_0| \\ -A, & 2XA \leq v_0|v_0| \end{cases}$$
$$v_t = \mathrm{sign}(a^*)\sqrt{Xa^* + v_0^2/2}$$
$$T_3 = \frac{1}{A}\sqrt{Xa^* + v_0^2/2}$$
$$T_2 = 0$$
$$T_1 = T_3 - v_0/a^*$$



If $|v_t| > V$ the velocity constraint is violated and the solution is not valid. Use Case 4.

- **Case 4: velocity saturated, acceleration saturated**

$$
\begin{aligned}
a^* &= A\,\mathrm{sign}(X) \\
v_t &= V\,\mathrm{sign}(X) \\
T_1 &= (v_t - v_0)/a^* \\
T_2 &= \frac{|X|}{V} - \frac{V}{A} + \frac{v_0^2}{2VA} \\
T_3 &= V/A
\end{aligned}
$$



Numerical considerations A few of the calculations can give unfeasible results in special cases if the numerical round-off errors are not taken into consideration.

We repeat the equations for solving Case 1 when $X/T_d \geq v_0/2$:

$$v_t = X/T_d + \sqrt{(X/T_d - v_0/2)^2 + (v_0/2)^2}, \quad X/T_d \geq v_0/2 \tag{7.58}$$

$$a^* = (2v_t - v_0)/T_d \tag{7.59}$$

$$T_1 = (v_t - v_0)/a^* \tag{7.60}$$

$$T_2 = 0 \tag{7.61}$$

$$T_3 = v_t/a^* \tag{7.62}$$

It can be shown algebraically that the value returned by (7.58) obeys

$$
\begin{aligned}
v_t &\geq v_0 \\
v_t &\geq 0
\end{aligned}
\tag{7.63}
$$

which is a requirement for (7.60) and (7.62) to return non-negative durations. Let δ be a number that can be resolved by the numerical representation used for computations, but δ^2 is rounded to zero. Now assume that $X/T_d = v_0/2 = \delta$. According to Eq. (7.58), the value of v_t is then

$$v_t = \delta + \sqrt{(\delta - \delta)^2 + \delta^2} \tag{7.64}$$

$$= \delta + \sqrt{0^2 + \delta^2} \tag{7.65}$$

$$= \delta + \delta \tag{7.66}$$

$$= 2\delta \tag{7.67}$$

However, since δ is small, the numerical solution will be

$$
\begin{aligned}
v_t &= \delta + \sqrt{(\delta - \delta)^2 + \delta^2} \\
&\approx \delta + \sqrt{0 + 0} \\
&= \delta \\
&< v_0 = 2\delta
\end{aligned}
\tag{7.68}
$$

thus violating Eq. (7.63). Similarly, we get $v_t \approx -\delta < 0$ if we use the input $X/T_d = v_0/2 = -\delta$. These errors become a problem when X and v_0 are close to zero and can be handled by adjusting the numerical output of (7.58) so that it always satisfies (7.63).

The same reasoning with opposite sign applies for $X/T_d \leq v_0/2$. For Case 3, similar reasoning leads to the need to verify that $v_t \geq v_0$ if $a^* > 0$ and $v_t \leq v_0$ if $a^* < 0$.

Strategy 2: Use Maximum Acceleration

This strategy applies the maximum acceleration to move the robot to the destination as fast as possible, hence, performing time-optimal trajectory generation. The motion will be jerkier than with Strategy 1, but the deadlines may be met more often. For each DOF, the expressions for a, T_1, T_2 and T_3 are first obtained by trying to solve Case 3 for Strategy 1, and if no solution exists, Case 4 is used. To make sure all DOFs reach their destinations simultaneously, the t_f value of the slowest DOF is then used as deadline and the algorithms of Strategy 1 are used to regenerate the trajectories.

7.4 Results

Simulations Based on Real Data

Figures 7.6 - 7.9 show simulated robot trajectories based on real data generated by the tracker described in Chapter 6. The simulations were performed with the assumption that the robot had two independent actuators that could move the tool in the x- and y-directions, respectively. The velocity was limited to $V = 2$ m/s and the acceleration was limited to $A = 20$ m/s^2, giving the robot properties similar to those of an ABB IRB 140 robot [ABB, 2013].

Figures 7.6 and 7.7 are based on the same tracker data, denoted data set 1, but in Fig. 7.6 the trajectory was generated using Strategy 1 and in Fig. 7.7 Strategy 2 was used. Similarly, Figs. 7.8 and 7.9 are based on the same data, denoted data set 2, but in Fig. 7.8 the trajectory was generated using Strategy 1 and in Fig. 7.9 Strategy 2 was used. The simulations serve to compare the strategy minimizing the acceleration with the time-optimal method.

In the upper part of each figure the target points estimated by the tracker are marked with blue stars. The stars are connected by blue lines in chronological order and the final target point is marked in red. The simulated robot trajectory is shown as a green curve. The positions the robot was at when new target points arrived are marked with green stars.

The lower left part of each figure shows the position, velocity and acceleration in the x-direction as functions of time, and the lower right part shows the corresponding functions in the y-direction. The final target position and time are marked by a star.

Comparing Fig. 7.6 and Fig. 7.7 it can be seen that Strategy 1 used smaller accelerations than Strategy 2 in the beginning and, hence, avoided making a detour to the first poor estimates of the target position. Due to a late update of the target position Strategy 1 was 0.4 mm away from the target at the deadline and reached the target 6 ms later. Strategy 2 was only 0.02 mm away from the target at the deadline and reached the target 1 ms later.

Comparing Fig. 7.8 and Fig. 7.9 it can again be seen that the trajectory generated by Strategy 2 made a larger detour to the first bad estimates of the target position. Both strategies reached the target in time and Strategy 2 even finished 7 ms before the deadline. Strategy 1, however, used less acceleration.

The Effect of Jerk on Resonant Systems

Simulations were performed to investigate how jerk limitation affects the reference tracking of a robot. The robot was modeled as a mass-spring-damper system, simulating how the difference between the position reference and the actual position of the robot arm was affected by the acceleration profile. The transfer function from the reference acceleration to the position error for the system used was

$$G(s) = \frac{1}{s^2 + 2\zeta\omega_0 s + \omega_0^2} \tag{7.69}$$

with $\omega_0 = 78.5$ rad/s and the relative damping $\zeta = 0.16$, which resulted in the amplitude being reduced by a factor of 3 in every oscillation period. The oscillation period was 81 ms, corresponding to a resonance frequency of 12.3 Hz. The parameters were chosen to resemble the properties of an ABB IRB 140 robot.

Figures. 7.10–7.14 show simulation results for three different acceleration profiles. Within each figure, all profiles moved the reference the same distance in the same time, having zero velocity and acceleration both in the initial point and in the destination point. The distance, however, varied between the figures. The blue curves show the result when the reference should be moved a given distance in minimum time with the jerk bounded by $J_{max} = 200$ m/s^3. The green curves show the results for moving the same distance in the same time with the jerk bound $J_{max} = 400$ m/s^3. The red curves show the result for the method presented in this chapter, with piecewise constant accelerations and infinite

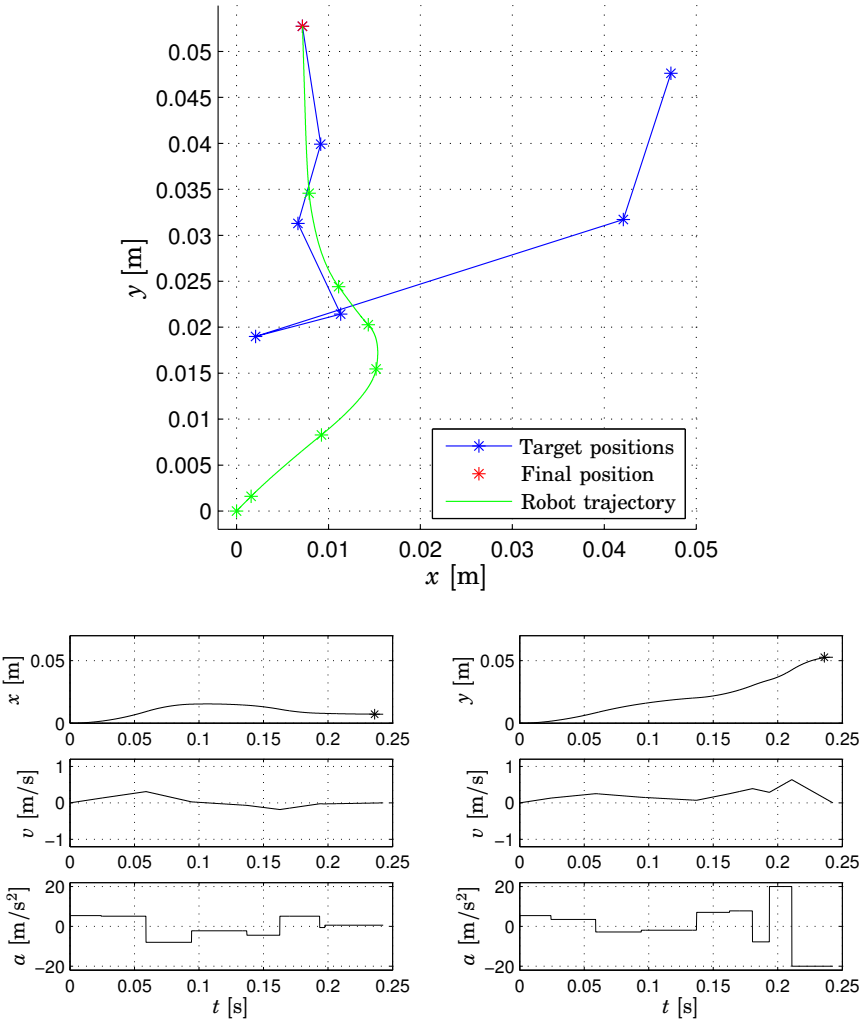


Figure 7.6 Generated trajectory based on data set 1, using trajectory generation method 1. (*Upper*) Target points and robot trajectory in the xy-plane. (*Lower left*) Time plots of position, velocity and acceleration in the x-direction. (*Lower right*) Time plots of position, velocity and acceleration in the y-direction.

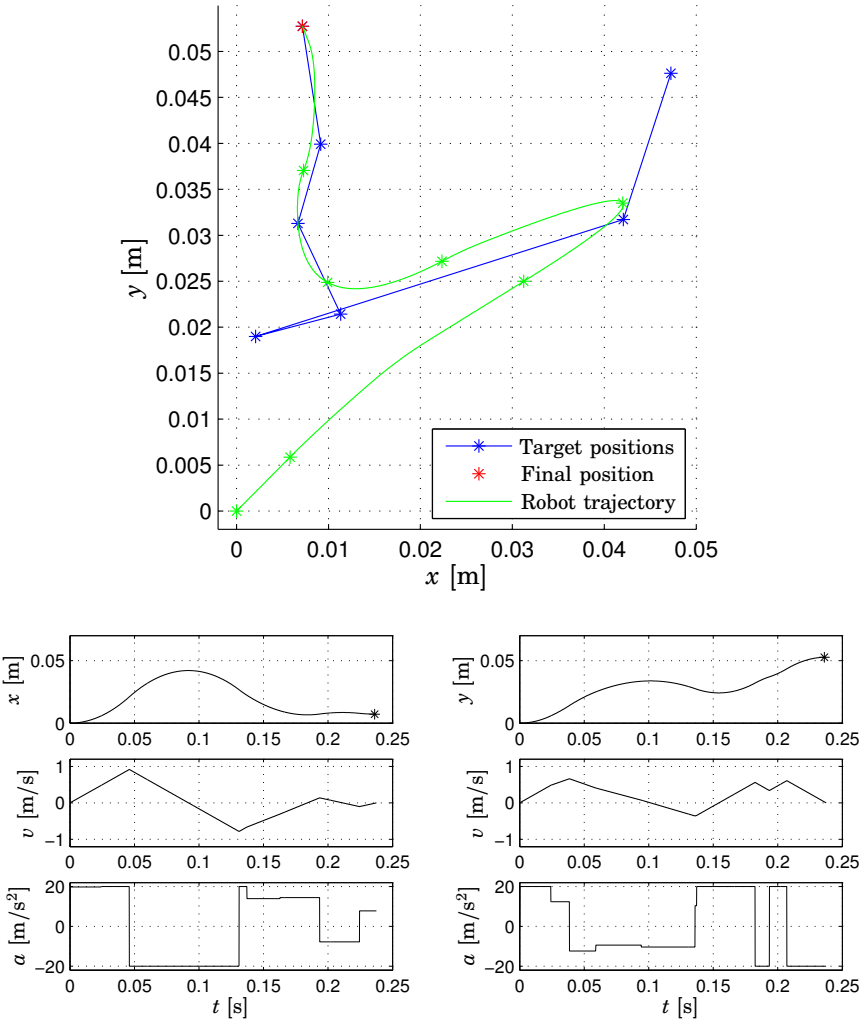


Figure 7.7 Generated trajectory based on data set 1, using trajectory generation method 2. (*Upper*) Target points and robot trajectory in the xy-plane. (*Lower left*) Time plots of position, velocity and acceleration in the x-direction. (*Lower right*) Time plots of position, velocity and acceleration in the y-direction.

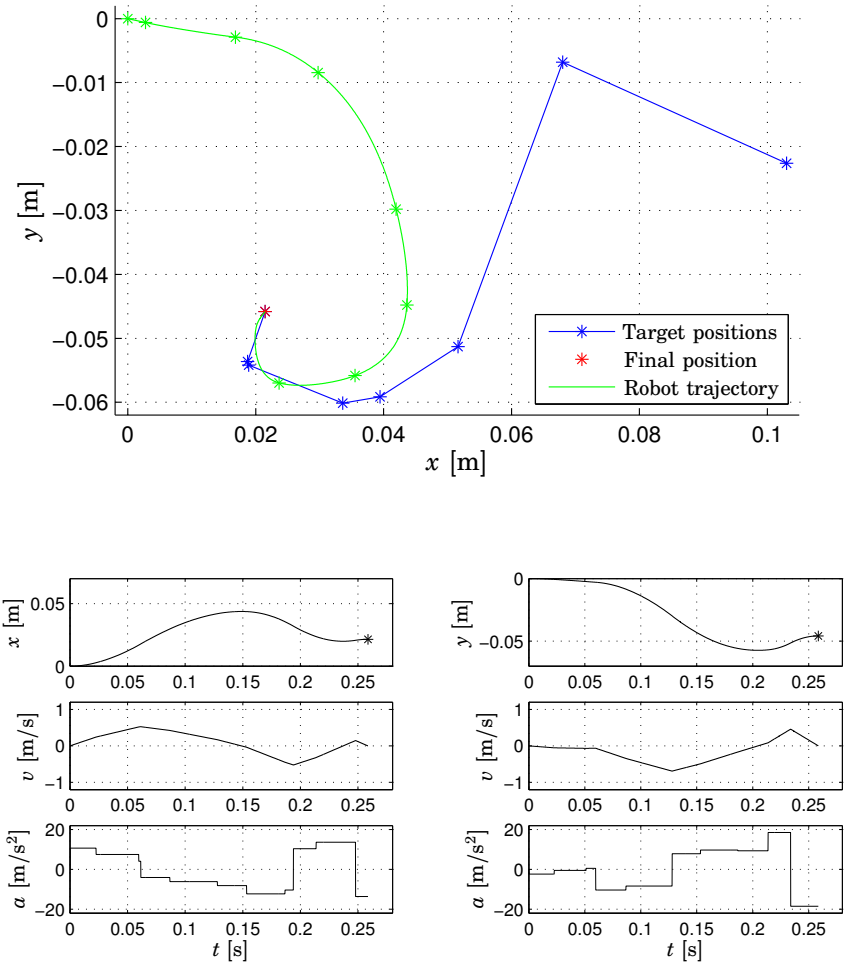


Figure 7.8 Generated trajectory based on data set 2, using trajectory generation method 1. (*Upper*) Target points and robot trajectory in the xy-plane. (*Lower left*) Time plots of position, velocity and acceleration in the x-direction. (*Lower right*) Time plots of position, velocity and acceleration in the y-direction.

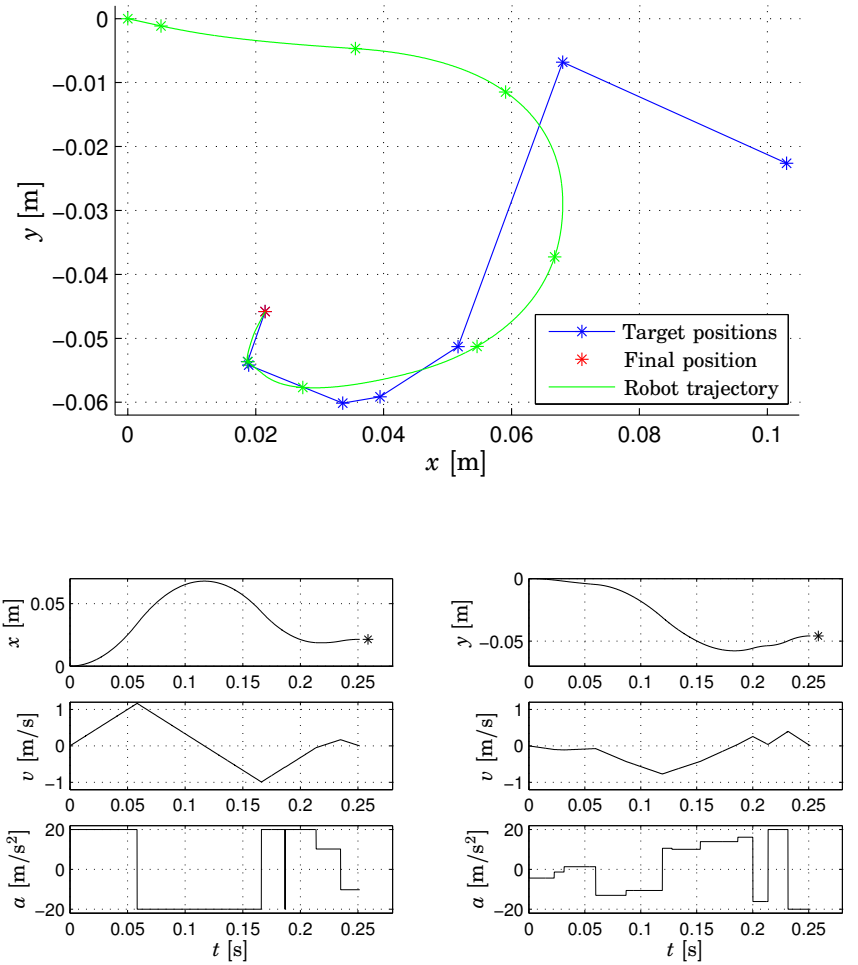


Figure 7.9 Generated trajectory based on data set 2, using trajectory generation method 2. (*Upper*) Target points and robot trajectory in the xy-plane. (*Lower left*) Time plots of position, velocity and acceleration in the x-direction. (*Lower right*) Time plots of position, velocity and acceleration in the y-direction.

jerk. The curves will be referred to as the J_{200}-curve, the J_{400}-curve, and the J_∞-curve respectively. For the different figures, the motions ranged from moving 1 mm in 54 ms to moving 3500 mm in 824 ms. The motion duration will be referred to as T_m. Repeated integration of the acceleration profiles gives that the total distance moved was

$$X = T_m^3 J/32 \tag{7.70}$$

with $J = 200$ m/s^3. The left column in each figure shows the reference profiles for acceleration, velocity and position. In the right column, the upper plot shows the position error. The two bottom plots show two different zoom levels of the reference position (dashed) the actual position of the robot link (solid).

The bottom right plot in each figure is the most interesting for judging the performance of a reference profile. It shows how quickly the actual position settled to the new reference position. In Fig. 7.10 with $T_m = 54$ ms the curves are very similar to eachother, but the error was the smallest for the J_∞-curve. The same qualitative result with more significant performance differences can be seen for $T_m = 117$ ms in Fig. 7.11. For $T_m = 200$ ms in Fig. 7.12 the J_{400}-curve and the J_∞-curve showed similar performance, which was significantly better than the J_{200}-curve. For $T_m = 252$ ms in Fig. 7.13 the J_{400}-curve and the J_∞-curve still showed similar performance, but now significantly worse than the J_{200}-curve. When T_m was increased further, as shown in Fig. 7.14, the J_{200}-curve had the smallest errors and the J_∞-curve had the largest errors. The trends can more easily be seen in Fig. 7.15, showing the position overshoot for the different acceleration profiles as a function of the motion duration. All accelerations, velocities and positions scale linearly with the jerk limitation. The relative performance between the acceleration profiles only depends on the duration of the motion. Figures 7.11–7.14 can, hence, be used to compare the performance of the different profiles when moving a constant distance with varying durations. The maximum jerk and acceleration would then be decreasing functions of T_m, where the maximum jerk can be extracted from Eq. (7.70):

$$J_{max} = 32X/T_m^3 \tag{7.71}$$

Figures 7.16–7.18 show simulations where the duration of the motion was different in the different figures, but the distance to move was constant. For the blue curves the jerk was limited to $J_{max} = 400$ m/s^3, while the green curves allowed infinite jerk. These simulations were designed to investigate how a constant jerk limitation performs compared to unlimited jerk when moving a constant distance with different motion durations. For $T_m = 150$ ms in Fig. 7.16, the bottom right plot shows that the J_∞-curve

behaved significantly better than the J_{400}-curve, similar to the result in Fig. 7.11. For $T_m = 200$ ms in Fig. 7.17 the J_{400}-curve had slightly smaller errors than the J_∞-curve, and for $T_m = 400$ ms in Fig. 7.18 the curves were almost indistinguishable. The trends can more easily be seen in Fig. 7.19, showing the position overshoot for the two acceleration profiles as a function of the motion duration.

Real-Time Execution

The trajectory generation methods were implemented in Java and used in real time to catch balls, as described in Chapter 8. The catching position used was the point where the trajectory of the ball intersected a vertical plane in front of the robot. For each of the 6 joints a trajectory was generated according to the following sequence every time a new target position was estimated by the tracker:

- Go to the target position before the deadline.
- Stay in the catching position for 0.5 s.
- Go to the home position in 1 s.
- Stay in the home position.

Usually only the first part of the trajectory was executed. When the catching position was updated by new measurements, a completely new trajectory was generated from the current state, and the old trajectory was discarded. This strategy can be considered to be a kind of MPC. Only the trajectory based on the final measurement of a throw was executed to the end. Since a safe return to the home position was generated every time a trajectory was generated, no special action had to be taken for the last estimate, and there was no risk for a trajectory to end while the robot was moving. The trajectory generation took approximately 0.1 ms on one of the cores of an Intel® Core™2 Quad CPU Q6600 processor running at 2.40 GHz. During that time, the algorithm described on pages 100–101 was executed in total 24 times; four times for each of the six joints according to the bullet list above. A single execution of the trajectory generation, hence, took approximately 4 µs.

7.5 Discussion

The presented method was easy to implement and could produce solutions in a fraction of a millisecond. Short execution times are good, but the achieved execution times were much shorter than is commonly required for robotic applications, and it can be argued that it may be worth

while to spend more time on trajectory generation to solve a more advanced optimization criterion, taking more properties of the system into consideration. The short computation times can be of larger benefit on embedded platforms with less computational power.

The presented method solves an optimization problem and only uses the first part of the solution. When new measurements become available, the remaining part of the old solution is discarded, and a new solution is computed. This makes the suggested approach a kind of MPC. Since the computation time of the optimization is independent of the duration of the motion, there is no reason to only calculate a solution for a receding time horizon.

Discontinuous acceleration profiles can excite resonances in mechanical systems, but the simulations in Figs. 7.10–7.19 show that they in some cases perform better than profiles with bounded jerk. The simulations used to generate Fig. 7.15, applying identical acceleration profile shapes for different motion durations, indicate that minimizing the acceleration and allowing infinite jerk gives good performance when the duration of the motion is similar to the period of the eigenfrequency of the resonant system, while minimizing the jerk gives less oscillations for motions with longer durations. The simulations used to generate Fig. 7.19 indicate that a constant jerk limitation has a very small effect on the performance if the motion does not require the maximum jerk to reach the destination in time.

The method presented in [Kröger and Wahl, 2010] may be considered to be more general than the method presented in this chapter, since it can generate trajectories with bounded jerk. However, when the specified duration of the motion is longer than the duration of the time-optimal trajectory, it applies the maximum jerk and acceleration, and minimizes the maximum velocity. In this sense it always applies the minimum allowed smoothness instead of maximizing the smoothness. One benefit of always using the maximum jerk and acceleration is that it maximizes the duration with constant velocity. This may be desirable for some applications, since the motion is linear while the velocities of all DOFs are constant.

In one sense, the method presented in this chapter optimizes the trajectories for smoothness, since it uses the smallest possible acceleration. Extending [Kröger and Wahl, 2010] to minimize jerk could be a way to generate even smoother trajectories that automatically sacrifice smoothness for speed when needed.

An alternative way to express that the trajectory generator minimizes the maximum acceleration, is to say that is minimizes the infinity norm of the acceleration.

One alternative for generating trajectories that do not excite the mechanical resonances, is to apply notch filters to the acceleration profiles,

removing the frequencies that match the eigenfrequencies [Erkorkmaz, 2004]. This may be feasible to do in real time, but it creates a few problems that have to be handled. If a notch filter is applied to a correct trajectory, the resulting trajectory may violate the velocity and acceleration constraints, and the total distance of the motion may be altered.

One natural extension of the method presented in this chapter would be to let it handle specification of non-zero velocities at the end of the trajectories.

In the section about real-time execution it was proposed to generate trajectories for just maintaining the current position. This may seem like a waste, but since the trajectory generation is so fast it may not be worthwhile to implement code for the special case of standing still. Another reason to not use the trajectory generation for standing still, is that it can run into numerical problems, as described on page 101. But enforcing the inequalities (7.63) handles the numerical problems and, it also avoids the need to choose a threshold specifying how short a motion must be for it to be considered as standing still.

The proposed method generates a trajectory to the target in the shortest possible time if the deadline cannot be met. This is a reasonable for the ball catcher, since the ball may arrive to the catching point later than expected, or future estimates of the catching point may be closer to the current position. For other applications it may be desirable to stop the motion and report that it was not possible to meet the deadline.

7.6 Conclusions

This chapter presented a method for generating trajectories, minimizing the maximum acceleration attained during the trajectory. Constraints on maximum velocity and maximum acceleration can be handled. It was also shown that trajectories allowing infinite jerk in some cases can cause less excitation of mechanical resonances than trajectories with limited jerk do.

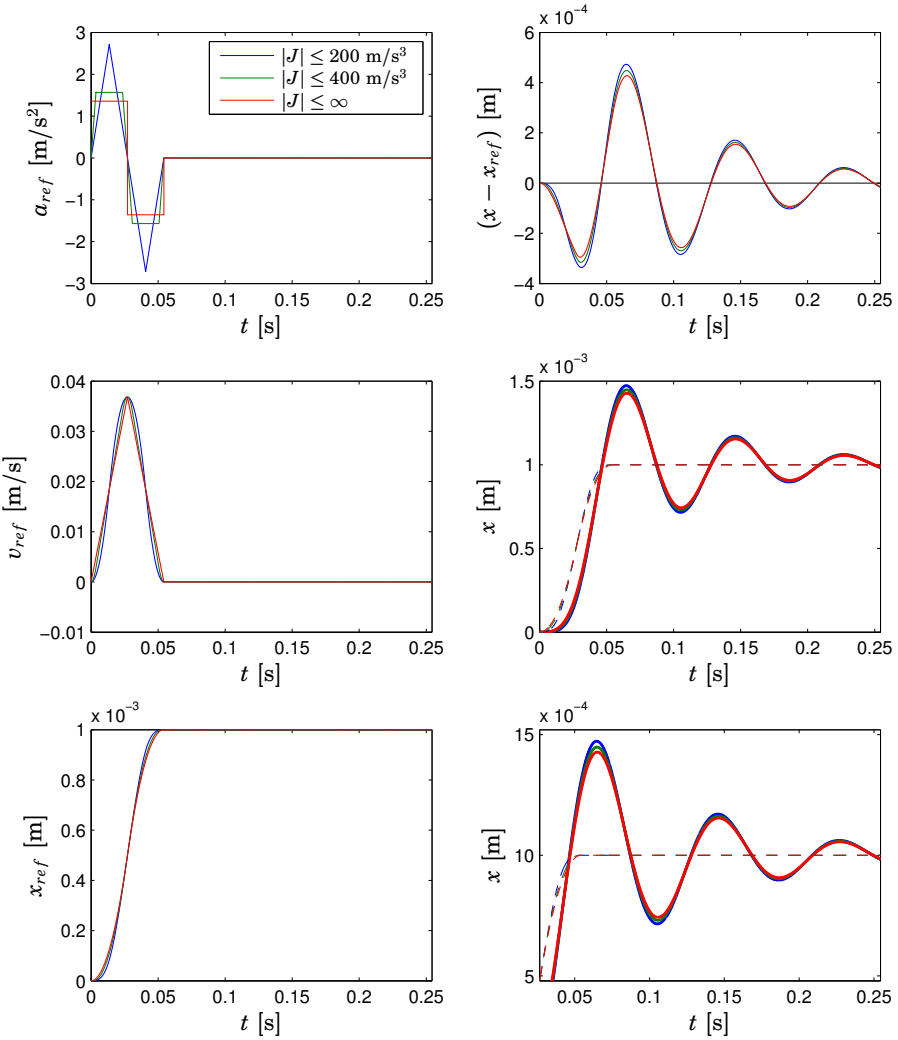


Figure 7.10 Simulated motion of the system (7.69) with different bounds on the jerk, J. In this figure the reference was moved **1 mm in 54 ms**.

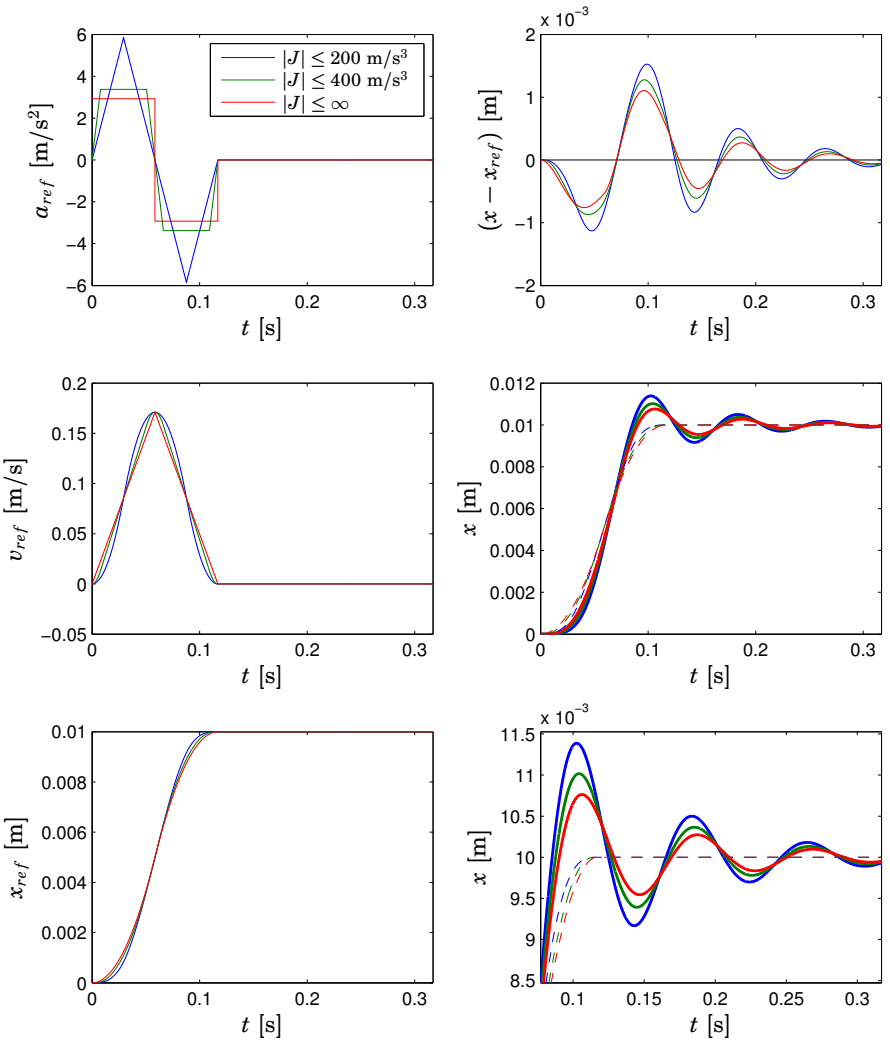


Figure 7.11 Simulated motion of the system (7.69) with different bounds on the jerk, J. In this figure the reference was moved **10 mm in 117 ms**.

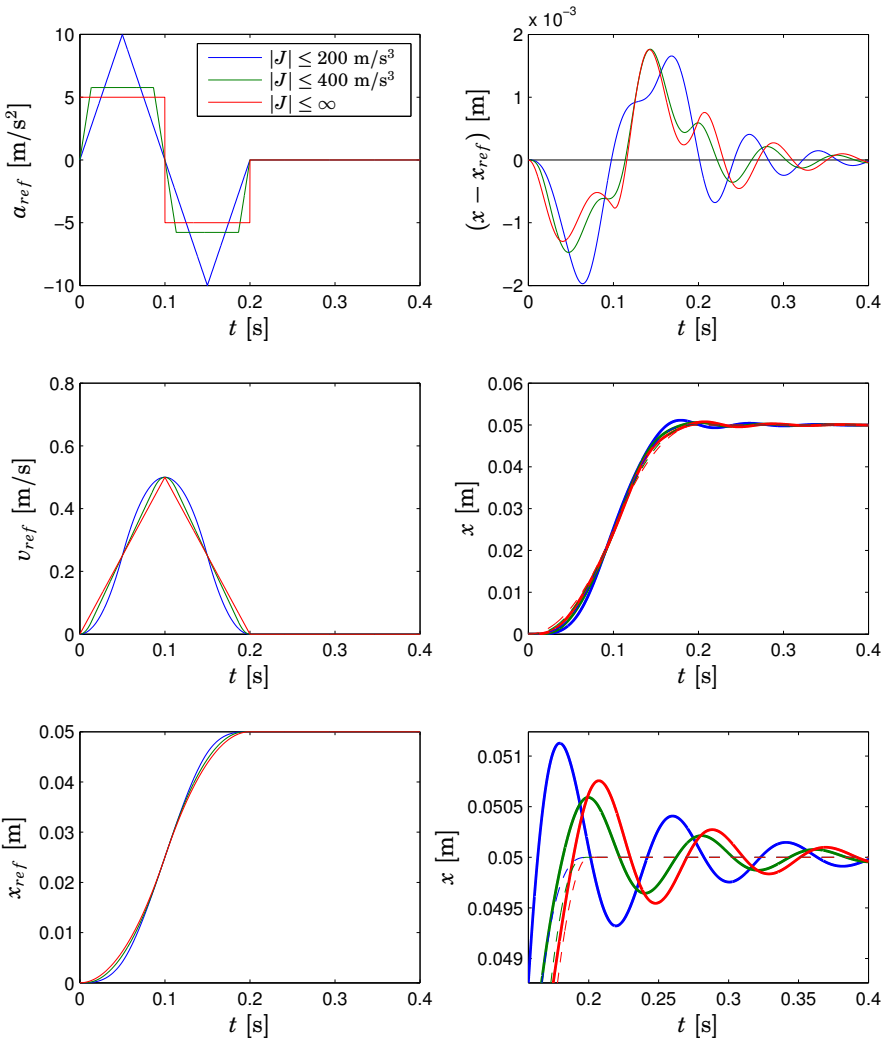


Figure 7.12 Simulated motion of the system (7.69) with different bounds on the jerk, J. In this figure the reference was moved **50 mm in 200 ms**.

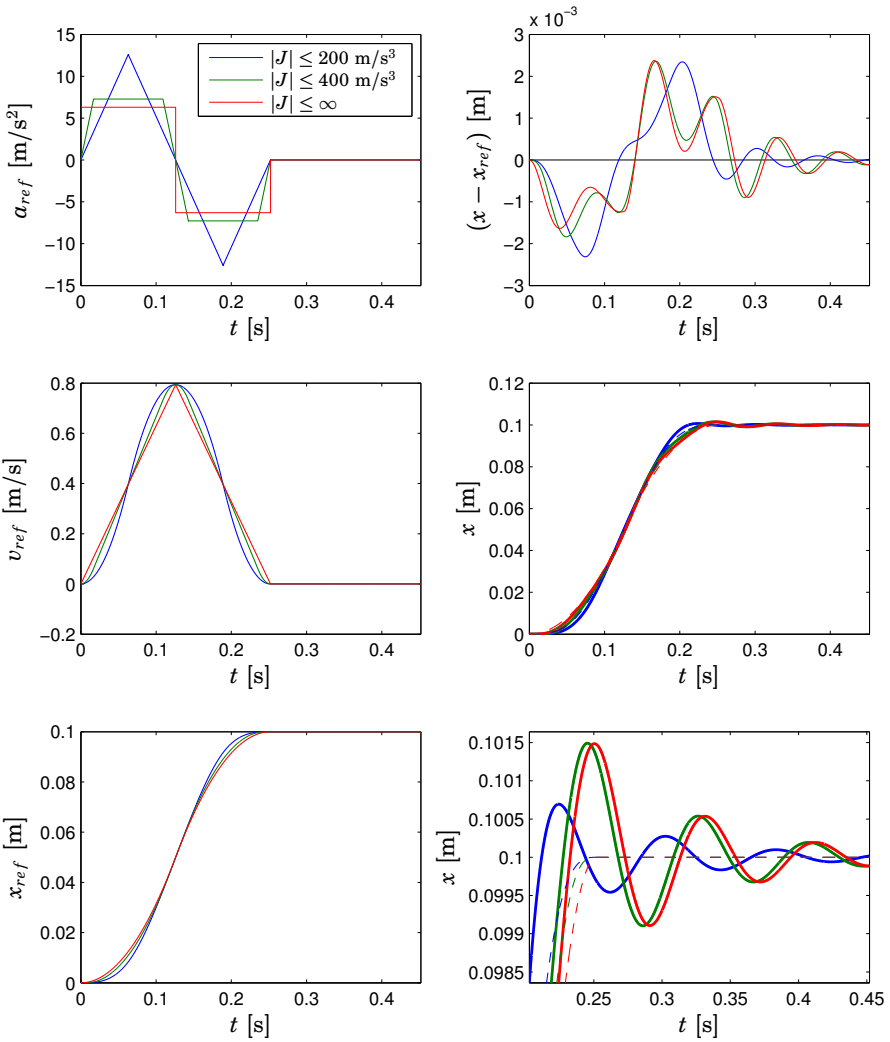


Figure 7.13 Simulated motion of the system (7.69) with different bounds on the jerk, J. In this figure the reference was moved **100 mm in 252 ms**.

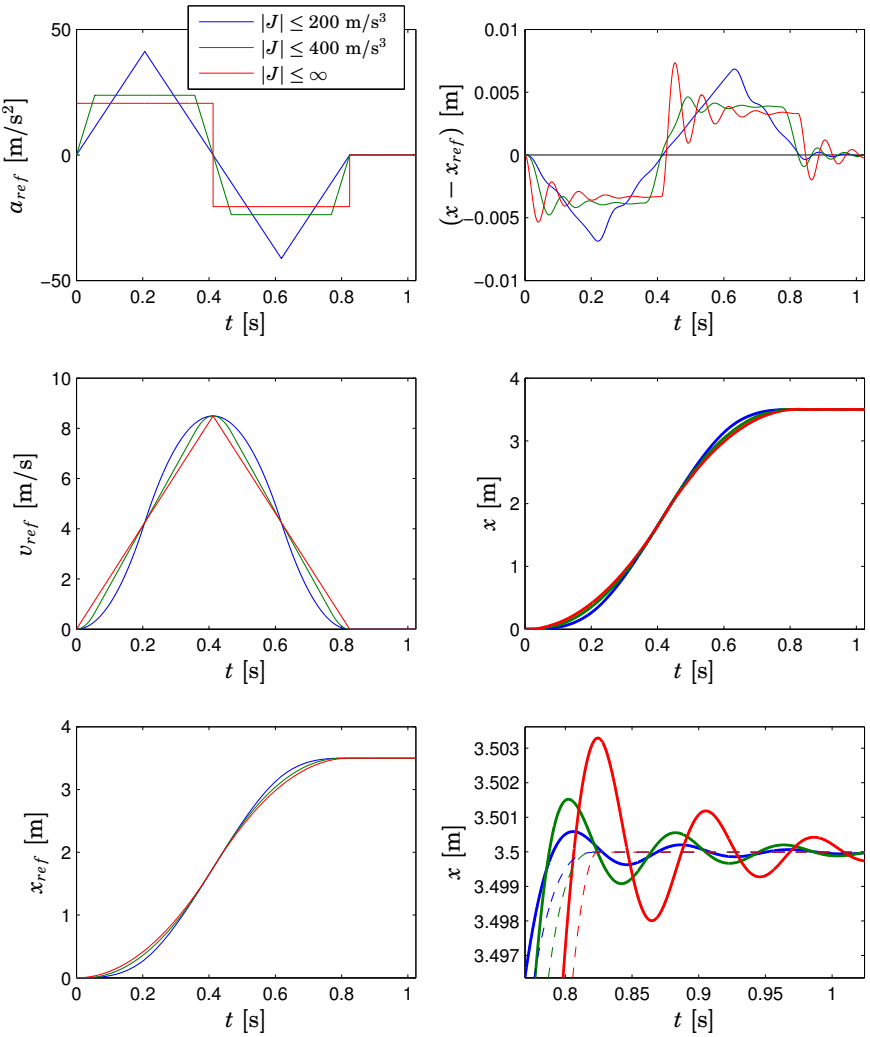


Figure 7.14 Simulated motion of the system (7.69) with different bounds on the jerk, J. In this figure the reference was moved **3500 mm in 824 ms**.

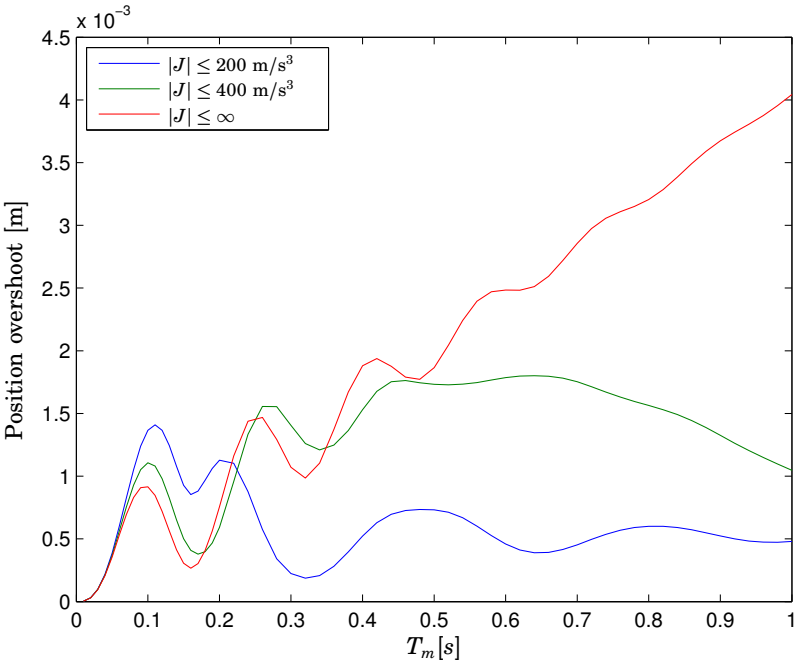


Figure 7.15 Position overshoots for acceleration profiles of the type presented in Figs. 7.10–7.14. The value of T_m is the duration from the beginning of the motion until the reference position reached its final value. Note that the distance traveled and the maximum acceleration are increasing functions of T_m for the acceleration profiles used to generate this figure.

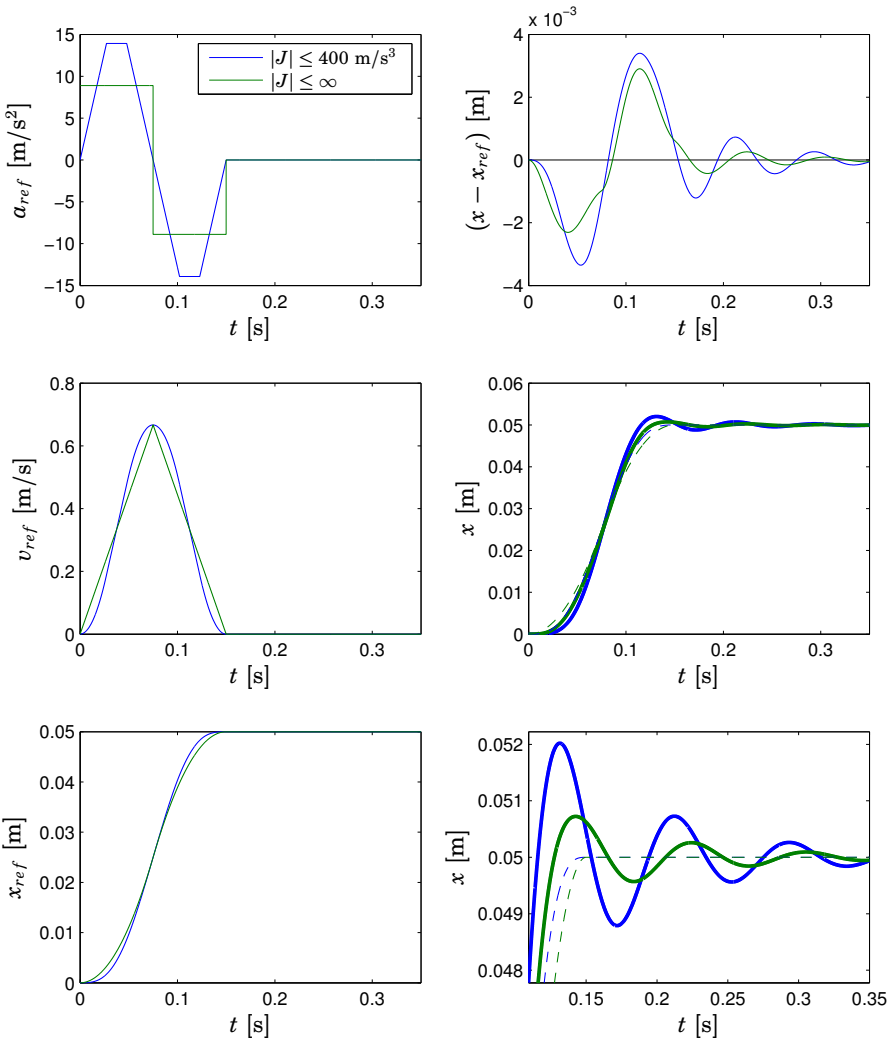


Figure 7.16 Simulated motion of the system (7.69) with and without bounds on the jerk, J. In this figure the reference was moved **50 mm in 150 ms**.

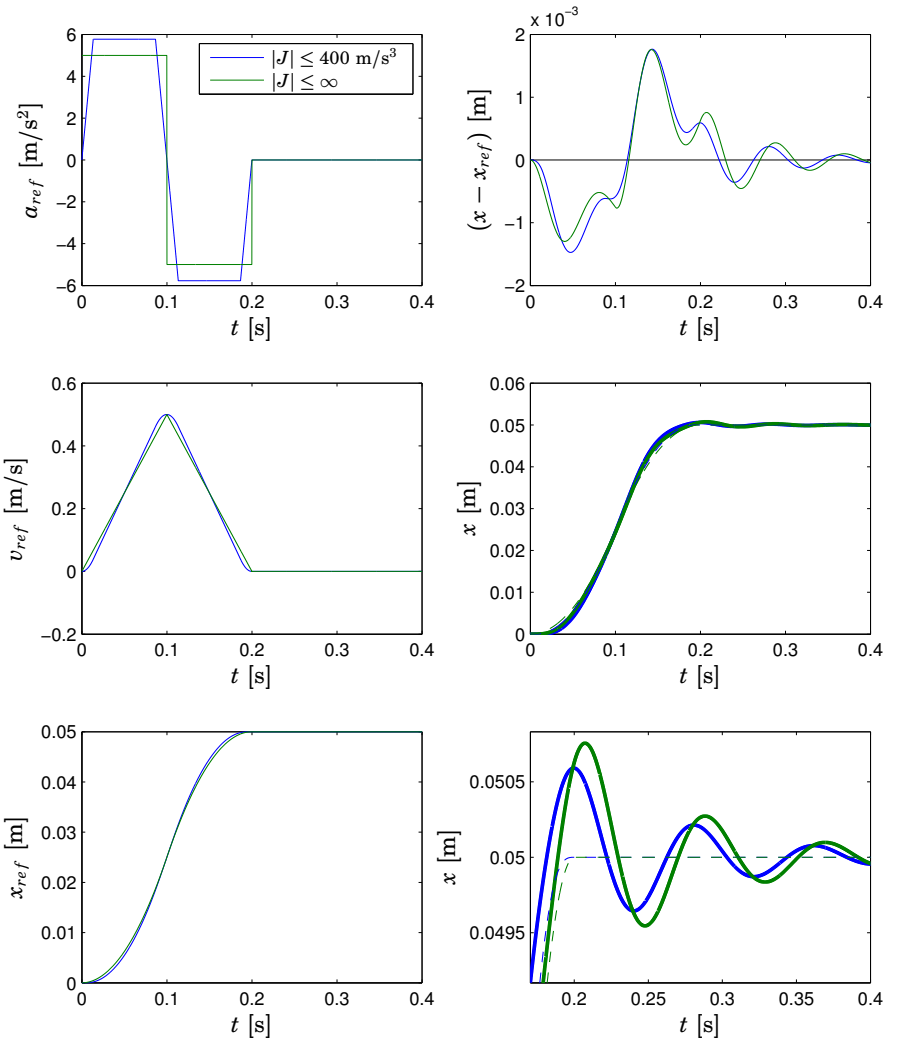


Figure 7.17 Simulated motion of the system (7.69) with and without bounds on the jerk, J. In this figure the reference was moved **50 mm in 200 ms**.

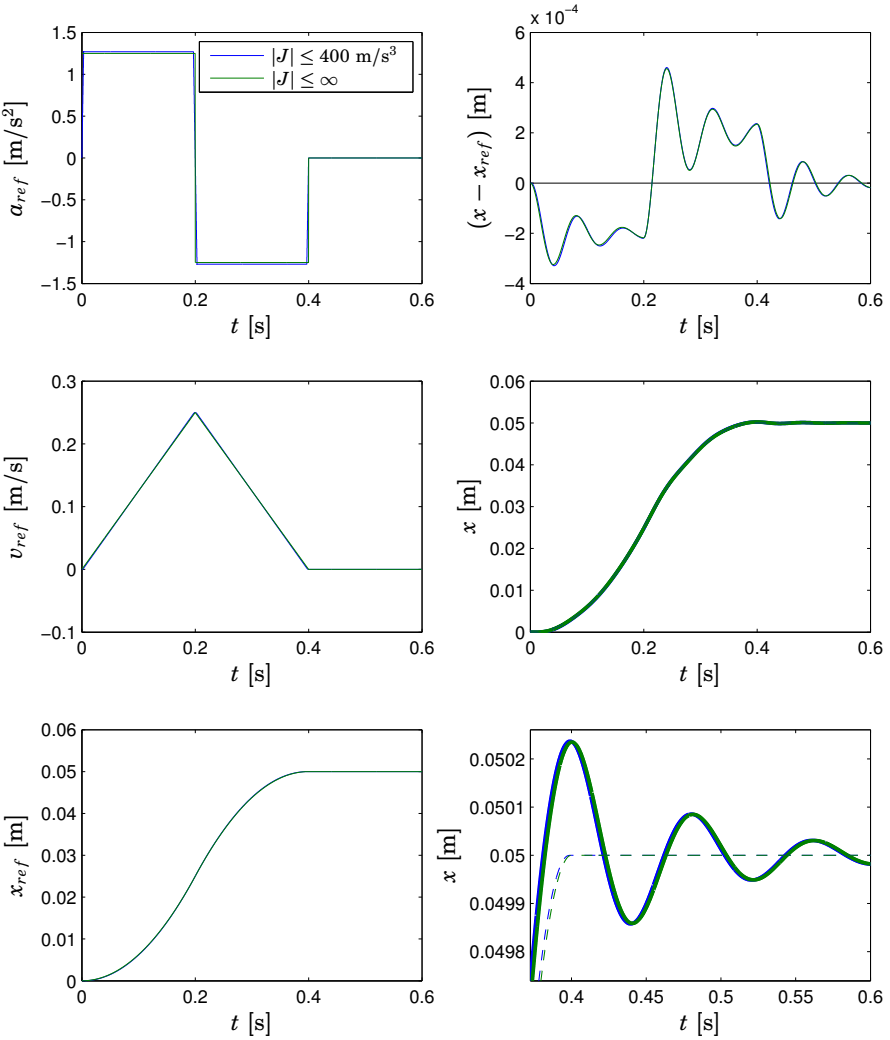


Figure 7.18 Simulated motion of the system (7.69) with and without bounds on the jerk, J. In this figure the reference was moved **50 mm in 400 ms**.

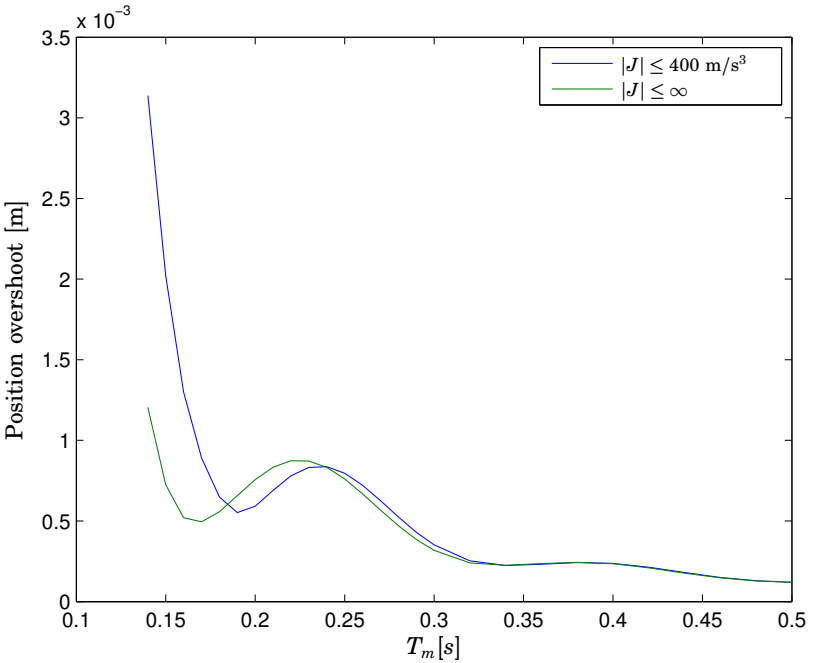


Figure 7.19 Position overshoots for acceleration profiles of the type presented in Figs. 7.16–7.18. The value of T_m is the duration from the beginning of the motion until the reference position reached its final value. The distance to travel was 50 mm.

8

System Integration of a Ball-Catching Robot

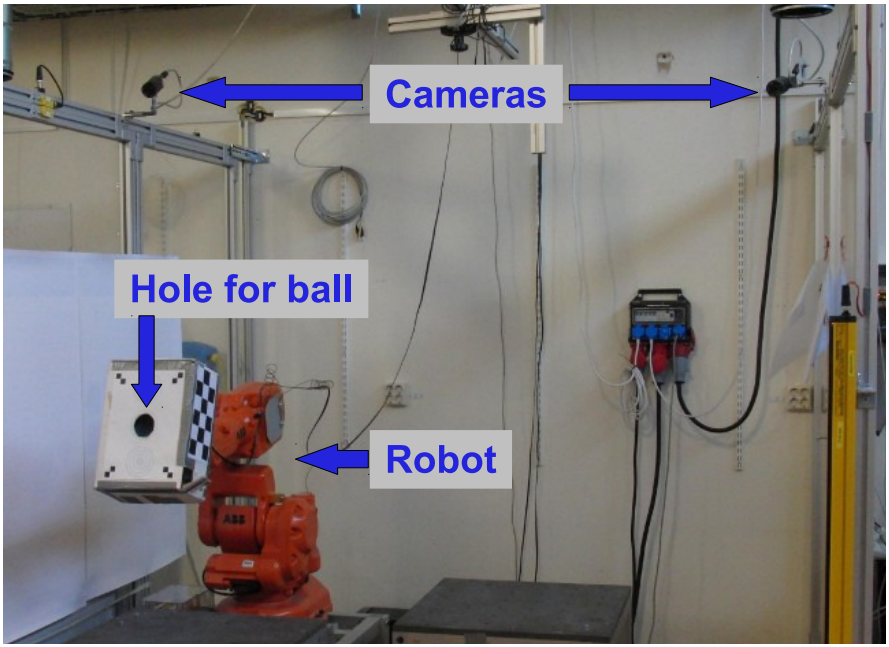


Figure 8.1 Experimental setup for ball-catching with an ABB IRB 140 robot and two cameras.

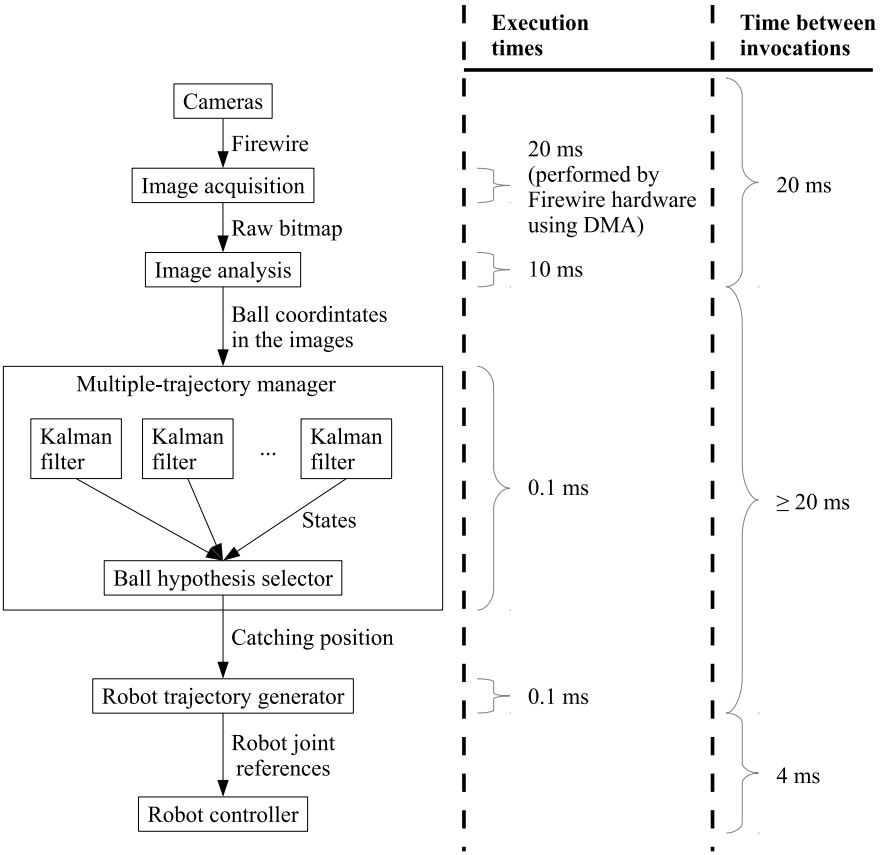


Figure 8.2 Overview of data flow, approximate execution times, and execution rates.

8.1 System Overview

This chapter describes how the research presented in Chapters 3–7 was used to create a ball-catching robot system, shown in Fig. 8.1. The first section gives an overview of the components and the data flow. More details about the different components are given in the following sections. A block diagram of the data flow is shown in Fig. 8.2.

Images were captured by two cameras with a frame rate of 50 FPS and transferred to the computer using a FireWire connection. A new pair of images was captured every 20 ms, and the transfer of the data to the computer took 20 ms. The computer was, hence, receiving data from the cameras without interruption. The FireWire hardware, however, used

DMA (Direct Memory Access) to write the image data to a ring buffer in RAM, and did not consume any CPU time of the computer.

Image analysis of the image pair took approximately 10 ms on a single core of the processor, and it returned the image coordinates of balls found in the images.

If any balls were found, their coordinates were sent to the tracker system, where the positions and velocities of the balls were estimated using EKFs (Extended Kalman Filters) [Schmidt, 1966]. Several hypotheses of ball trajectories were maintained, partly to handle falsely detected balls from the image analysis, and partly to be able to track several balls simultaneously. Every measurement was either matched with an existing trajectory hypothesis and used for a Kalman filter measurement update, or the measurement was used to create a new trajectory hypothesis. Then the set of hypotheses was pruned to remove unlikely hypotheses. The remaining trajectory hypotheses were evaluated based on the number of inlier measurements and the direction of flight. If any hypothesis was good enough, the best one was used to calculate the catching position, which was passed to the robot trajectory generator. Finally, a Kalman filter time update was performed on all trajectory hypotheses. Executing the tracker took approximately 0.1 ms.

The trajectories were designed to have piecewise constant acceleration, minimizing the magnitude of the acceleration needed to reach the catching position before the ball got there. Every time a new estimate of the catching position was generated by the ball tracker, a new trajectory for the robot was generated based on its current position and velocity. Generation of the robot trajectory for all joints, including the trajectory back to the home position after the ball was caught, took approximately 0.1 ms. The trajectories were represented as piecewise second-order polynomials. Every 4 ms they were sampled, and reference positions and velocities were sent to the robot controller.

All computations were performed on a desktop computer with an Intel® Core™2 Quad CPU Q6600 processor running at 2.40 GHz.

The balls used for throwing had a diameter of 60 mm and weighed approximately 21 g. The catching device used was a cardboard box with a hole. The diameter of the hole was chosen so the ball had to be squeezed through the hole.

8.2 Image Analysis

The task of the image analysis was to identify foreground objects and discriminate them from the background. In the context of the ball catcher, the foreground consisted of green flying balls, and everything else was

background.

Cameras

For image acquisition, two cameras of the type Basler A602fc were used, capturing images with 656×480 pixels. The cameras were run at 50 FPS (frames per second) with 4 ms exposure time.

In the coordinate system of Fig. 2.2, the focal points of the cameras were placed at the coordinates $(X, Y, Z) = (1.65, 1.00, -1.03)$ m and $(X, Y, Z) = (-0.44, 1.12, -0.95)$ m, i.e., on the left and right sides of the robot, slightly behind and above it, see Fig. 8.1. The cameras were pointing toward the area in front of the robot, where the thrower should stand. This positioning gave a large area of stereo coverage in front of the robot. The maximum distance between the robot and the positions where the ball could be detected was mainly limited by the resolution of the cameras. Having the cameras next to the robot allowed the system to detect an incoming ball when it was far away, and get the best accuracy when the ball was close to the robot (and the cameras). This setup was well suited for a system where the robot started moving to an approximate catching position as soon as the ball was detected and the catching position was continuously refined based on the new measurements, with the highest requirement on the accuracy just before the catching instant.

The cameras were calibrated using the Camera Calibration Toolbox for Matlab available through [Bouguet, 2010]. Markers on the catching box were used to measure a camera's position relative to the box, and the relative position between cameras was determined by capturing images of a checkerboard pattern in different poses visible by both cameras [Linderoth, 2008].

Data Transfer

Data transfer from the cameras to the computer was done with the IEEE 1394 serial bus, more commonly known as FireWire or iLink, using the libdc1394 C library [Douxchamps, 2012]. The cameras were connected to two FireWire ports, which in turn were connected to a single FireWire root node in the computer. The data was transferred as 8-bit raw Bayer-pattern images at a rate of 50 FPS.

The frame rate was limited by the data rate through the FireWire root node. If a computer with two FireWire root nodes was used, images could be transferred to the computer at 100 FPS.

The images were transferred to the computer as raw Bayer-pattern images, where each pixel only had information about one color component, as depicted in Fig. 8.3. Demosaicing was performed using the "simple method" in libdc1394, taking 2.8 ms for the image pair. This demosaic-

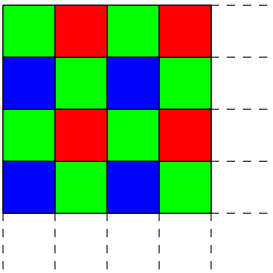


Figure 8.3 Using a Bayer pattern, each pixel only records light from the red, green or blue color channel, as illustrated in the figure. In the process of *demosaicing* the colors are interpolated to create full RGB images with values for all three color channels for all pixels.

ing method resulted in poorly reproduced colors near sharp edges in the images, but these artifacts only had a small impact on the ball-detector algorithms, and more advanced demosaicing algorithms would take too long to execute.

It should be noted that the execution times reported in Chapters 3 and 4 include the 1.4 ms needed to demosaic each Bayer-pattern image.

The cameras also provided the possibility to do the demosaicing in the camera hardware and transfer the image in YUV(4:2:2) format (16 bits/pixel average), but this would require twice as much data to be sent over the FireWire connections, halving the achievable frame rate, and this option was hence rejected. More details about the image formats can be found in the camera reference manual [Basler, 2005].

Pixel Classification

After demosaicing of the Bayer-pattern images, the next step of the image analysis was to classify each pixel based on its color. This process generated an image, where the value of each pixel was its estimated probability of belonging to the foreground, hereafter denoted the *probability image*. More details on how the classification was done is given in Chapter 3, and examples of classified images are shown in Fig. 3.7. The classifier was designed to handle varying illumination conditions that were combinations of daylight and fluorescent light.

In many images of a moving ball, significant motion blur was visible, which reduced the accuracy of the classifier in Chapter 3. The method presented in Chapter 4 was used for reducing the effects of the motion blur, making it possible to measure the positions of the balls more accurately.

Since the cameras were not moving, and moving balls should be detected in front of a fairly static background, there was much to be gained

from comparing the most recent image to a background image. A common way to obtain a background image is to somehow create the median of a set of images. I instead used one of the most recent images as background, which was possible since the balls moved so fast that the projections of a ball in two consecutive images rarely overlapped. Using a recent image as background also had the advantage that it adjusted very quickly to changes in the background or illumination, and it did not require any extra processing.

Ball Detection

When the probability of belonging to the foreground had been computed for each pixel, the next step was to detect the balls. To this purpose *pixel scores* were calculated by subtracting 0.5 from the probability image. The scores were, hence, positive for pixels that were most likely to be part of a ball, and they were negative for pixels that were most likely to be background.

The positions and sizes of the balls were estimated by locally maximizing the *circle score*, which was defined as the sum of the pixel scores enclosed by a circle. Decreasing the radius from its optimum value would exclude pixels with positive scores from the circle, and increasing the radius from its optimum would include pixels with negative scores in the circle. Moving the center of the circle from its optimum would reduce the circle score due to both of the above mentioned effects. Examples of detection results can be seen in Figs. 4.5(b) and 4.5(c).

To find balls in an image, the approximate position was first estimated by looking for areas having many pixels with positive scores. This operation was performed using integral images and a binary-fashion search. The entire image was split in two halves, and the half with the most foreground pixels was in turn split in half. This procedure was repeated recursively until an area that mostly contained foreground pixels was found.

When the approximate position and size of a ball had been estimated, the local maximum of the circle score was found using a gradient-based search. To facilitate fast computations, only integer values were used for the center coordinates and the radius of the circle. The optimal circle was then estimated with sub-pixel resolution by doing a second order approximation of the circle score around the best set of integer parameters.

The Effect of Flicker

Many electric lights, such as fluorescent lights commonly used in indoor environments, flicker at twice the frequency of the AC voltage powering

them. In Europe the power grid has a frequency of 50 Hz, resulting in light flicker at 100 Hz, i.e., the flicker has a period of 10 ms.

The exposure time used for the ball catcher was 4 ms, and since this was significantly lower than the period of the flicker, the intensity of the captured images depended on the phase angle of the flicker at the time of the exposure. Furthermore, the images were captured with a period of 20 ms. Since this was a multiple of the period of the flicker, aliasing effects were expected if the actual frequencies were not exactly equal to those specified.

Let f_s be the frame rate of the camera (the sampling frequency), f_{ac} the frequency of the AC voltage, and $f_f = 2f_{ac}$ the frequency of the flicker. Under the assumptions $f_s \approx 50$ Hz and $f_{AC} \approx 50$ Hz, the aliased frequency observed in the images is then

$$\begin{aligned} f_a &= |((f_f + f_s/2) \mod f_s) - f_s/2| \\ &= 2\,|f_{AC} - f_s| \end{aligned} \tag{8.1}$$

Measurements on the power outlets in the lab showed frequencies in the range 50 ± 0.08 Hz, which according to (8.1) corresponded to aliased frequencies with periods down to 6.25 s, assuming that $f_s = 50$ Hz. These results corresponded well to what was observed in the images, where the illumination intensity appeared to oscillate with varying periods down to approximately 6 s.

The aliasing effects observed in the images had implications on the image analysis. Since the motion detection was based on changes in intensity of the different color components, varying illumination of the scene could cause falsely detected motions. Since one of the recently captured images (40 ms old) was used as background, and the intensity variation caused by the aliasing varied slowly, there was no practical impact on the motion detection.

If the illumination of the scene was a combination of daylight and fluorescent light, the aliasing also caused variations in the observed colors. The pixel-classification method described in Chapter 3 is, however, robust to this kind of variation and handled the disturbance without any further action.

8.3 Tracking

The non-linear state-space model (2.4) was used to model the flight of the ball. The sampling interval was set to $h = 20$ ms to match the frame rate of the cameras. The air-drag parameter, c, was estimated by recording data from a number of throws made with various speeds and directions and then minimizing the one-step prediction errors, resulting in the value

$c = 0.04\ \mathrm{m}^{-1}$. The method described in Chapter 6 was used to track the balls.

8.4 Robot Control

Robot and Interfaces

The robot used for the ball catcher was an ABB IRB 140 [ABB, 2013], shown in Fig. 8.4. It had a maximum payload of 6 kg, reach of 810 mm and position repeatability of ± 0.03 mm.

The robot was controlled by an IRC5 controller extended with an external controller through the ExtCtrl interface [Blomdell et al., 2005; Blomdell et al., 2010]. It connected to the low-level joint controllers of the robot with a sample rate of 250 Hz and let the higher level functionality be executed in an external controller. For each joint the external controller could set the reference position, and provide feedforward data for the velocity and motor torque. Measured positions and velocities, commanded motor torques, etc. were sent back from the robot control cabinet to the external controller.

The data was transferred using the LabComm protocol [LabComm, 2013], which allowed the specification of data types that should be sent over a socket. The communication overhead has been kept to a minimum and the protocol is appropriate for sending samples of process data in real time.

Trajectory Generation

For the ball-catching application, only the destination points of the trajectories were important, and the paths to the destination were of little interest. Hence the trajectory generation was performed in joint space. This gave limited control of the paths compared to trajectory generation in task space, but for the assigned range of catching positions the paths were always well behaved.

The joint trajectories were designed to have piecewise constant accelerations, and to use the smallest possible acceleration to reach the target before the deadline, given the constraints on maximum acceleration and speed. Every time a new estimate of the catching position became available, a new robot trajectory, based on the current position and velocity, was generated in approximately 0.1 ms. More details on the trajectory generation can be found in Chapter 7.

The IRB 140 has 6 joints, where the three outer joints form a spherical wrist. For the ball catcher, however, joints 4 and 6 were kept at fixed angles, resulting in a robot that effectively had 4 joints, as illustrated by the skeleton in Fig. 8.4. Joint 1 had a vertical rotation axis, and joints 2,

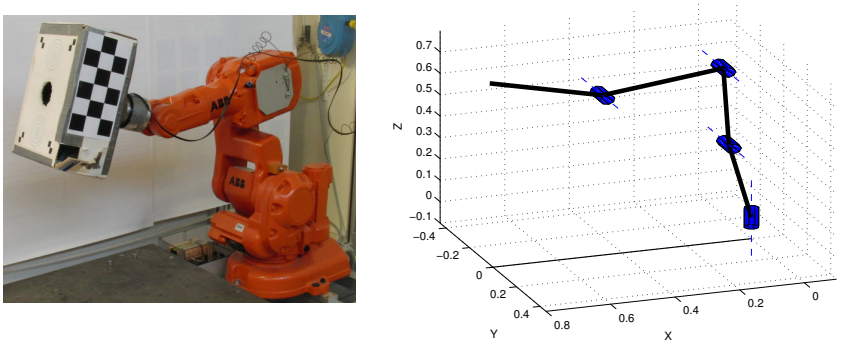


Figure 8.4 Photo of the robot and a sketch of the used joints.

3, and 5 all had horizontal rotation axes perpendicular to the plane that intersected all joint centers. This configuration allowed positioning in 3D and reorientation around joint 5.

To make the catch as easy as possible, it was desirable to reorient the box so that the velocity vector of the ball at the catching instant was normal to the face of the box, but with only one rotational DOF, such a reorientation was usually not possible. The normal vector of the face of the box was confined to lie in the plane that intersected the centers of all robot joints. Hence, the normal vector was chosen as the projection of the negative velocity vector into this plane, thus minimizing the angle between the negative velocity vector and the normal of the box.

If the ball was thrown from a position in front of the robot (the area covered by the cameras' field of view) toward the work space of the robot, the horizontal direction of the ball was quite limited. The direction of the velocity of the ball mainly varied due to different velocities in the Y- and Z-directions, which could be accounted for by joint 5 of the robot. Hence, locking joints 4 and 6 had a very small effect on the performance of the ball catcher.

In a general trajectory-generation context it could be desirable to generate trajectories with non-zero end-point velocities. For the chosen way of catching the ball–by means of a box with a hole–there was no need for such functionality.

The catching positions were confined to be in the plane $Z = 0$ in the range $-0.25\text{ m} \leq X \leq 0.55\text{ m}$ and $-0.20\text{ m} \leq Y \leq 0.55\text{ m}$. The asymmetric X range was due to the presence of a wall next to the robot, cf. Figs. 2.2 and 8.1.

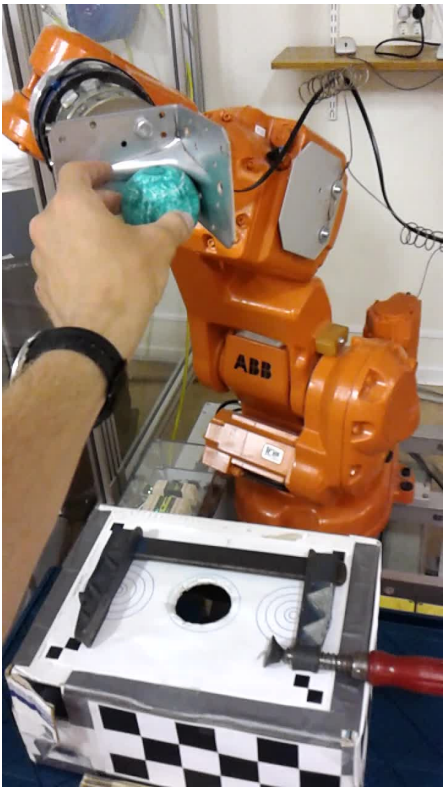


Figure 8.5 The experimental setup used to estimate the position accuracy required for the ball to pass through the hole.

8.5 Accuracy Requirement

Figure 8.5 shows an experimental setup that was used to measure how accurately the catching position had to be estimated in order to accomplish a successful catch. The box was put in a fixed position with its face in the horizontal plane, as shown in the bottom of Fig. 8.5. The robot was equipped with a tool with a corner that the ball could be pressed against, as shown in the top part of Fig. 8.5. This setup could be used to drop the ball from well-defined positions. A set of experiments was performed, where the center of the ball always was 450 mm above the face of the box when the ball was dropped, but various positions in the horizontal plane were used. For each dropping position, 10 experiments were performed, and the number of balls that passed through the hole of the box was recorded. The results can be seen in Fig. 8.6.

The outcome of the experiments was affected by the required accuracy

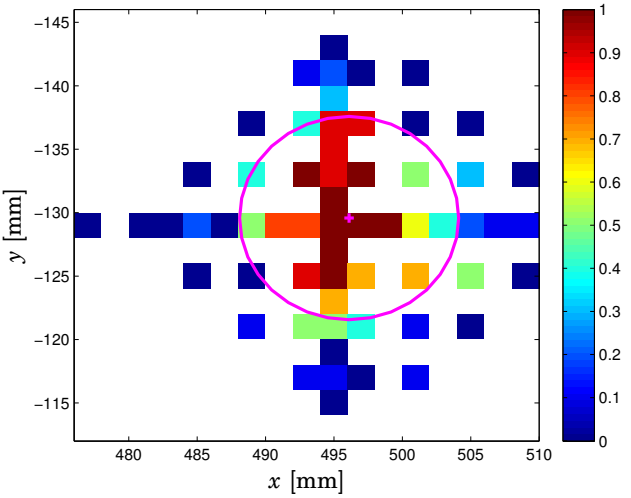


Figure 8.6 Results from experiments of the type illustrated in Fig. 8.5. For each position indicated by a non-white square, the ball was dropped 10 times from an elevation 450 mm above the face of the box. The color of the square indicates what fraction of these experiments passed through the hole.

to enter the hole, but also by how much the position of the ball varied after being dropped from the corner on the robot tool. The model used was that if the center of the ball deviated less than the distance r from the center of the hole, the ball always passed through the hole, and if it deviated more from the center of the hole, the ball always bounced out. The actual position where the ball hit the box was assumed to be normally distributed with the covariance matrix $\sigma^2 I_{2\times 2}$ and the mean value straight under the position where the ball was dropped from the robot. The values of r, σ, and the center coordinates of the hole were estimated by the maximum-likelihood estimate, i.e., the set of values that was the most likely to generate the data in Fig. 8.6. The computations were performed by means of numerical optimization and gave the result $r = 8$ mm (marked by a magenta circle in Fig. 8.6) and $\sigma = 7$ mm.

The position of the box during the experiments was constant, but its absolute position was not known. The offset of the axes and the center coordinate of the circle in Fig. 8.6 were, hence, irrelevant. The important data were the value of r and relative distance between the dropping positions.

Since the ball entered the hole in the straight downward direction

during these experiments, the gravity could help to pull the ball through the hole if it bounced a little on the edges of the hole. In a real catching situation, where the face of the box was closer to vertical, the required accuracy was likely to be slightly higher.

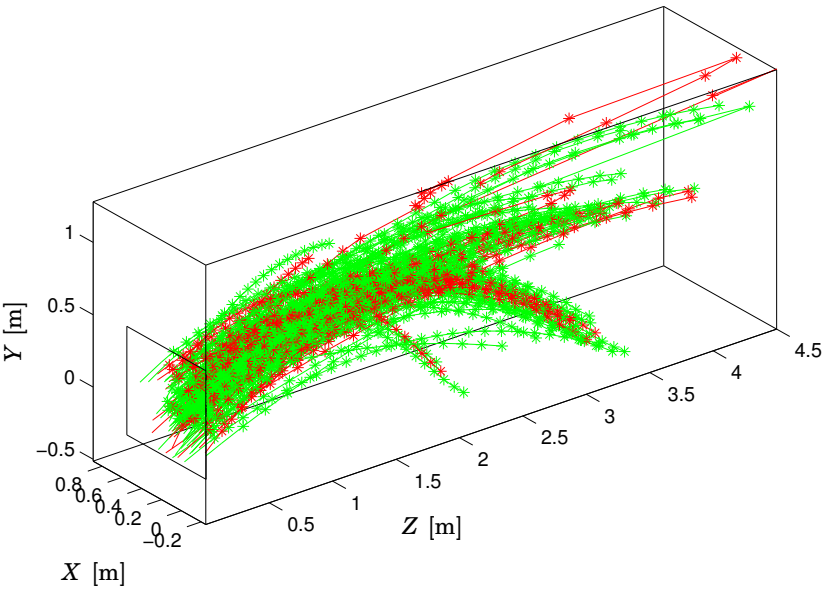


Figure 8.7 Estimated trajectories of the throws that were used to evaluate the catch rate of the robot. The trajectories that resulted in a successful catch are marked in green, and the failures are marked in red. The black rectangular frame in the plane $Z = 0$ indicates the designated catching range.

8.6 Performance

To evaluate the performance of the ball catcher, an experiment with a number of throws toward the robot was performed. Out of the 108 throws that entered the catching range of the robot, 78 were caught, giving a success rate of 72 %. The estimated trajectories of the throws are plotted in Fig. 8.7. The throws were performed from distances in the range 2 – 7 m from the robot, but the balls were usually not detected further away from the robot than 4 m. The flight durations were in the range 0.35 – 0.90 s

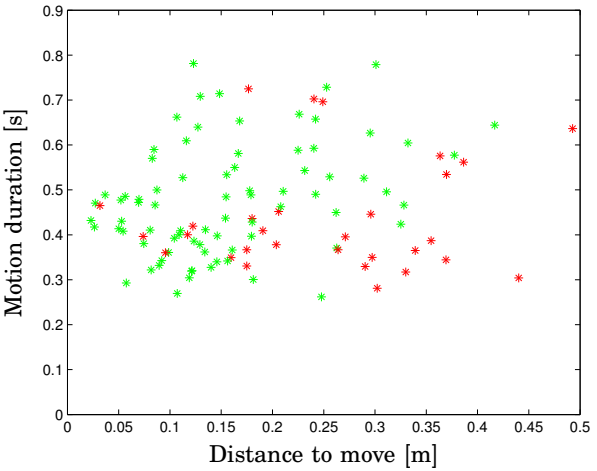


Figure 8.8 Plot of motion duration vs. the distance required to move to catch the ball. Green: successful catch. Red: failure.

and the horizontal speeds in the range 3 – 11 m/s. The angles between the velocity vector of the ball and the horizontal plane at the catching instant were in the range 5 – 45 degrees.

The 60-mm ball had to be squeezed through the hole of the catching box, but due to the rounded shape of the ball and the compliance of the box, the ball would enter the hole if the position of the hole deviated less than 8 mm from a perfect catch, as described in Sec. 8.5.

Figure 8.8 shows the duration from the instant when the first robot trajectory was generated until the estimated catching instant on the vertical axis, and the distance the robot had to move on the horizontal axis. The large number of failures in the lower right part of the figure, where the distance to move was large in comparison to the available time, indicates that the robot acceleration was an important (but not the only) limiting factor for the success rate.

Out of the ball catchers developed by other groups, the systems presented in [Bäuml et al., 2011a] and [Lippiello and Ruggiero, 2012a] are the most notable. In [Bäuml et al., 2011a], the radius of the opening in the hand was 2 cm larger than the radius of the ball. The system achieved a success rate of 80 % for throws made 5 – 7 m away from the robot with durations of 0.6 – 0.9 s. In [Lippiello and Ruggiero, 2012a], the size of the hand allowed position errors of 5 cm. The ball was thrown from distances of 6 – 6.5 m away from the robot and the flight duration was 0.7 – 0.9 s. They achieved a catch rate of 70 %.

To conclude, my system demonstrated higher position accuracy in the catching pose than the other ball-catching robots. It also showed a much faster reaction time, being able to catch balls that had a short time of flight.

8.7 Discussion and Future Work

The presented implementation determined the catching position by calculating where the trajectory of the ball intersected a vertical plane in front of the robot. The catching range could probably be increased by choosing a catching surface that was better adapted to the kinematics of the robot, e.g., a torus. The choice of catching point could be generalized further by, for example, choosing the point along the ball trajectory that the robot could reach the fastest.

In the presented system the ball was caught using a box with a hole in, whereas several other similar systems have used robotic hands or similar gripping devices. Catching with a hand demonstrates higher temporal accuracy, but the hand may add a significant cost to the system and be prone to damages by the impact of the ball. The hands have also been used to form a basket, allowing spatial uncertainties of a few cm. This is a reasonable design choice if the objective is to maximize the catch rate using a reasonably-sized hand. Using a box with a hole was a very cheap and robust solution, and by choosing a small diameter of the hole it demonstrated very high spatial resolution of the ball catcher.

When looking the performance of different ball catchers, it should be noted that many systems have special challenging requirements that make them hard to compare directly. In [Bäuml et al., 2011a] the catching was performed with a humanoid robot with a highly compliant upper body and on-board sensing, making the estimation difficult due to shaking cameras etc. In [Lippiello and Ruggiero, 2012a], only a single camera was used. I [Bätz et al., 2010], very accurate timing and velocity tracking was required, since a basket ball should be caught on a flat plate without bouncing off.

If the orientation of the catching pose would have 2 DOFs, it would be possible to always make the velocity vector of the ball perpendicular to the face of the box. This was, however, not practical, due to the vicinity of a kinematic singularity in the wrist of the robot, and in practice it was usually possible to make the face of the box almost perpendicular to the velocity vector. It could also be possible to mount the box on the robot in a different way, avoiding the kinematic singularity.

The cameras were placed on the cage of the robot cell, see Fig. 8.1, largely to interfere as little as possible with other activities in the robot

lab. This resulted in a rather wide base line of 2.10 m, helping to get accurate depth information about the ball. The positioning on the cage also meant that the cameras were somewhat distant from the catching position. If the cameras were closer to the robot (and the ball), the measurements of the ball could be more accurate, since a given error in the image would result in a smaller error in 3D space. Having the cameras mounted in fixed positions made it possible to calibrate them with high precision. If the cameras could be moved, however, they could cover the entire flight of the ball even if they were given a narrower field of view. This could be used to adjust the zoom level (which could be constant or varying during execution) and make better use of the resolution of the cameras.

Every image was analyzed in its entirety. When an estimate of a ball's trajectory was available, time could be saved by only looking for the ball around its predicted position in the images, but this could reduce the robustness of the detector. For example, it would be impossible to track multiple balls simultaneously, and an estimated ball trajectory based on false measurements could prevent detection of a ball that was actually present in the image. Further, the image processing time was approximately 10 ms, and with the frame rate used the image processing could take almost 20 ms. Hence, there was no big incentive to reduce the image processing time at the cost of reduced quality of the analysis result.

The two images captured simultaneously were processed on a single CPU core, one after the other. Since the images were analyzed completely independently, it should be easy to let them be processed on one core each, probably giving almost half the processing time. The image analysis algorithm was highly parallelizable, since more than 75 % of the processing time was spent on operations on a single pixel at a time or on groups of 2×2 pixels. This could be used to reduce the image processing time further on a system with many CPU cores or a GPU.

The aliasing effect caused by the flickering lights could be used to measure time drift between the cameras and detect offsets in the synchronization. The offset could be estimated by looking at the phase shift in the oscillation of the illumination. However, since the frequency of the AC voltage could be either less than or greater than the frame rate, it would be difficult to determine which camera was ahead of the other. It was, hence, difficult to measure the offset and compensate for it, but the flickering could be used to detect whether the cameras are out of synch.

8.8 Conclusions

This chapter described the integration of several components to form a ball-catching robot system and discussed different practical aspects. The system demonstrated higher spatial accuracy and shorter response time than ball-catching robots developed by other groups.

Part II

Force Control and Estimation

9

Robotic Assembly

9.1 Introduction

The traditional way of programming industrial robots is to use position-based control and make the robots follow predefined trajectories. Modern robots do this well with high speed and accuracy, and they are widely used in industry, e.g., for welding, spray painting, and pick-and-place operations. There are, however, limitations to what can be achieved with position-based control strategies. For the accuracy of the robots to be exploited, they have to work in structured environments, where the positions of all objects are well known, which often is a feasible requirement in a factory setting.

In many assembly applications, however, the accuracy of the positioning in fixtures and grippers is not enough for a position-based approach to work. Then it may be beneficial to introduce force sensing. If the contact forces can be detected, it is possible to infer the relative position of the objects to be assembled, without knowing exactly how they are gripped. Equipping the robot with this extra *sense* makes it possible to approach the ability of the human hand to perform complicated assembly operations in spite of poor position accuracy. One should also remember that, even when it is possible to position all work objects accurately, it may be desirable to avoid the job it requires, by using a force-controlled strategy.

There are different ways of using force sensing in robotic assembly. One simple strategy is to use the sensor as a detector of contact events. This can be useful if the robot should make contact with an object that has an uncertain position. The robot then searches for the object by moving toward it, and when the measured contact force exceeds a specified threshold, a contact event is detected, typically triggering the start of the next motion in the assembly operation. Similarly, the force sensor can be used to detect unexpected contacts (collisions) and trigger error messages. In the above example a simple force threshold was used, but filtering techniques can be used to detect more complex force signatures.

When contact has been made, as described in the previous paragraph, it is common that the next step in the assembly operation is a sliding operation along the contact surface. If the normal direction of the contact surface is known and there is some compliance in the system (e.g., in the robot, the gripper, or the work object), the sliding can be performed using position control. If there are large uncertainties, or if the robot and the involved parts are very stiff, a position-based strategy for sliding may, however, lead to lost contact or very large contact forces that can damage the equipment. In this situation it may be necessary to use continuous force feedback to control the contact force. The benefit of this strategy is that the contact forces can be kept at acceptable levels without knowledge about the shape of the contact surface, but this strategy also requires a more skilled programmer, since it may be difficult to find good feedback parameters for the force control.

Using force control also puts new demands on the task specification. Programming languages for industrial robots typically have no or limited support for force-controlled operations and require an extension in order to perform force-controlled assembly.

In [Bruyninckx et al., 2001] a survey of the requirements for autonomous robotic assembly was presented, along with an overview of functionality accomplished, and directions for future work. On the lowest level contacts can be managed by passive compliance [Lane, 1980] or active control strategies based on force sensing, such as hybrid position/force control [Raibert and Craig, 1981] or impedance control [Hogan, 1985]. Approaches for specifying the end-effector behavior include the task frame formalism [Mason, 1981], the operational space formulation [Khatib, 1987], and the constraint-based task-specification framework [De Schutter et al., 2007]. On a higher level, the assembly tasks require sequencing [Fox and Kempf, 1985; Wang et al., 1998]. For robust assembly it is also important to detect and recover from errors [Di Lello et al., 2012; Nägele et al., 2012].

Numerous application examples exist. In [Jörg et al., 2000] a combination of vision and force control was used to insert pistons into a moving engine block. The force sensor used had built-in compliance, which reduced the bandwidth required for the force control. The use of sensorimotor primitives for task specification was proposed in [Morrow et al., 1995]. The method was demonstrated by inserting cable connectors into their sockets by means of vision and force sensing.

The work presented in this chapter has been performed to further investigate the possibility of using robots for automated small-parts assembly. In order to be industrially interesting, they must be competitive in comparison to the current state of the art, i.e., human workers. Hence, the productivity of a human assembly worker has been used as a benchmark of the performance.

Two assembly scenarios have been used. One example was the attachment of a shield can onto the PCB (printed circuit board) of a mobile phone. The involved parts were small and delicate, and the shield can should be squeezed onto its socket, thus requiring compliance and detection of contacts to fit it properly. The other example was the assembly of an emergency stop button. It consisted of slightly larger components that could withstand larger forces, but it required many different types of contact operations for the complete assembly, including peg-in-hole insertion, snap-fit attachment, and putting a nut on a thread and screwing it until it was tightened.

Section 9.2 describes the framework that was developed for specifying and executing the assembly tasks. Sections 9.3 and 9.4 then describe two use-case assembly tasks, followed by experimental results in Sec. 9.5, discussion in Sec. 9.6, and conclusions in Sec. 9.7.

9.2 Task Specification and Control Framework

This section describes the principles used for task specification and the control framework that was developed to execute the tasks.

The tasks were hierarchically subdivided into *skills* and *motions*. A typical *motion* could be to move to a given position, or to search for a contact force along a given direction. Motions were combined to form *skills*, which could typically be to pick up an object, or to assemble two objects as a part of the full assembly *task*.

Specification of Force-Controlled Motions

The specification of the motions using force sensing was done with a framework based on iTaSC (instantaneous Task Specification using Constraints) [De Schutter et al., 2007]. An overview of the parts relevant for this chapter is given on next three pages.

The iTaSC methodology provides a convenient way of describing robot motions by imposing constraints (e.g., on positions, velocities, or forces) without being restricted to joint space or Cartesian space. One or several *kinematic chains* are defined, relating the tool frame of the robot to the world frame. Each kinematic chain can be seen as an alternative description of the 6-DOF pose of the tool frame, parameterized by a vector, χ_f, of 6 *feature coordinates*. A kinematic chain consists of a series of transformations, e.g., translations and rotations, and it is typically designed so the feature coordinates represent quantities relevant for the task.

For convenience, object and feature frames can be introduced to facilitate the specification of the kinematic chain. The relations between these

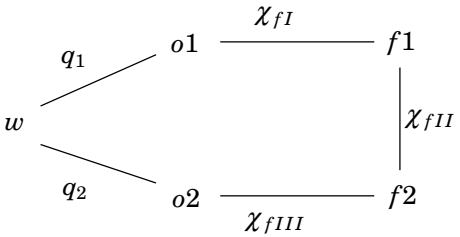


Figure 9.1 Illustration of how the frames are connected by the joint coordinates, q, and the feature coordinates, χ_f.

frames and the world frame, w, are depicted in Fig. 9.1, and below is a description of the frames and guidelines for how to choose them.

- $o1$ – object frame 1. Usually attached to the object to manipulate;
- $f1$ – feature frame 1. Usually attached to a feature on the object to manipulate;
- $f2$ – feature frame 2. Usually attached to a feature on the robot;
- $o2$ – object frame 2. Usually attached to the robot.

The feature coordinate vector, χ_f, can be partitioned into three parts between the different frames according to Fig. 9.1 with

$$\chi_f = ({\chi_{fI}}^T \quad {\chi_{fII}}^T \quad {\chi_{fIII}}^T)^T. \tag{9.1}$$

The six DOFs of χ_f can be distributed between χ_{fI}, χ_{fII} and χ_{fIII} as appropriate for the task at hand. Similarly, for two-arm operations, it is convenient to partition the robot joint coordinates as $q = ({q_1}^T \quad {q_2}^T)^T$, where q_1 and q_2 each represents the joint coordinates for one arm.

The quantities to be constrained are chosen by specifying outputs:

$$y = f(q, \chi_f). \tag{9.2}$$

The outputs can be functions of the joint coordinates and the feature coordinates, but if the kinematic chains are chosen properly, each output can often be made to directly correspond to one of the feature coordinates.

The fact that the transformations in Fig. 9.1 form a kinematic loop, imposes a constraint on the relation between the joint coordinates and the feature coordinates, which can be expressed as

$$l(q, \chi_f) = 0, \tag{9.3}$$

where the function l depends on the kinematic chains chosen for the task. This relation can also be expressed in terms of the individual transformations as a constraint on the total transformation matrix:

$$T_w^{o1}(q_1)T_{o1}^{f1}(\chi_{fI})T_{f1}^{f2}(\chi_{fII})T_{f2}^{o2}(\chi_{fIII})T_{o2}^{w}(q_2) = I_{4\times4}, \tag{9.4}$$

where T_a^b denotes the transformation from frame a to frame b.

Control The control is performed on the velocity level. Consequently, a relation between the time derivatives of the different variables is needed. Taking the time derivative of the output equation (9.2) gives

$$\dot{y} = C_q\dot{q} + C_f\dot{\chi}_f, \tag{9.5}$$

where $C_q = \partial f/\partial q$ and $C_f = \partial f/\partial\chi_f$. Similarly, taking the time derivative of the kinematic loop constraint (9.3) gives

$$J_q\dot{q} + J_f\dot{\chi}_f = 0 \Longleftrightarrow \tag{9.6}$$

$$\dot{\chi}_f = -J_f^{-1}J_q\dot{q}, \tag{9.7}$$

where $J_q = \partial l/\partial q$ and $J_f = \partial l/\partial\chi_f$. Substituting (9.7) into (9.5) now gives

$$\dot{y} = A\dot{q}, \tag{9.8}$$

$$A = C_q - C_fJ_f^{-1}J_q \tag{9.9}$$

Equation (9.8), hence, relates the outputs, y, (the signals one wants to control) to the joint coordinates, q, (the signals that can be commanded).

The robot motion is described in terms of the outputs on the velocity level:

$$\dot{y}_d^\circ = \dot{y}_d + C \tag{9.10}$$

where $\dot{y}_d$ is a feedforward velocity for the desired trajectory and C is a controller using feedback from sensors, e.g., force sensors, cameras or measured robot joint positions. The feedforward and the control output are added to form $\dot{y}_d^\circ$; the reference output derivative that is used to calculate the commands to the robot. The desired robot joint velocities, $\dot{q}_d$, can then be obtained as a solution to

$$\dot{y}_d^\circ = A\dot{q}_d \tag{9.11}$$

Unless y and q have equal dimensions and A has full rank, some kind of pseudoinverse of A has to be used to solve (9.11). For underconstrained systems one possibility is

$$\dot{q}_d = A^\#\dot{y}_d^\circ, \qquad A^\# = M^{-1}A^T\left(AM^{-1}A^T\right)^{-1} \tag{9.12}$$

which is the solution to the optimization problem

$$\begin{aligned} &\min_{\dot{q}_d} \dot{q}_d^T M \dot{q}_d \\ &\text{s.t.} \quad \dot{y}_d^\circ = A\dot{q}_d \end{aligned} \tag{9.13}$$

The matrix M is a weighting matrix that is used to choose the optimization criterion. If the desired weighting is more easily described in Cartesian space, then this property can easily be handled by using $M = J_q^T(q)\bar{M}J_q(q)$, where $\bar{M}$ describes the weights in the Cartesian space.

Since the above equations work on the velocity level, $\dot{q}_d$ from (9.12) has to be integrated w.r.t. time to provide q_d. The generated q_d and $\dot{q}_d$ are used as reference trajectories. Low-level joint controllers of the robot are assumed to make the robot track these references.

Calculation of Feature Coordinates

In order to calculate (9.8), both q and χ_f must be known, but only q can be measured directly. The corresponding feature coordinates, χ_f, that solve (9.4) can be obtained iteratively, which corresponds to calculating the inverse kinematics of a robot. For the purpose of finding χ_f, q can be considered to be constant, and the left hand side of (9.4) can be expressed as $T_q(\chi_f)$, which should satisfy

$$T_q(\chi_f) = I \tag{9.14}$$

Assume that there is an initial guess $\hat{\chi}_f^0$ of χ_f. If $\hat{\chi}_f^i \neq \chi_f$ this results in an error, ΔT^i, satisfying

$$T_q(\hat{\chi}_f^i) = \Delta T^i, \tag{9.15}$$

$$\Delta T^i = \begin{bmatrix} \Delta R^i & \Delta t^i \\ 0 & 1 \end{bmatrix} \tag{9.16}$$

where ΔR^i is a rotation matrix, which can be converted to a rotation vector Δv^i in axis/angle representation [Spong et al., 2006]. Using a linear approximation around $\hat{\chi}_f^i$, the error of the feature coordinates, $\Delta\chi_f^i$, is given by

$$J_f(\hat{\chi}_f^i)\Delta\chi_f^i = \begin{bmatrix} \Delta t^i \\ \Delta v^i \end{bmatrix} \tag{9.17}$$

The feature coordinates χ_f can then be obtained iteratively with

$$\hat{\chi}_f^{i+1} = \hat{\chi}_f^i - \Delta\chi_f^i = \hat{\chi}_f^i - J_f^{-1}(\hat{\chi}_f^i) \begin{bmatrix} \Delta t^i \\ \Delta v^i \end{bmatrix}, \tag{9.18}$$

which can be seen as a special case of the Gauss-Newton algorithm.

If the system were linear, a single iteration would be needed to arrive at the correct solution. If χ_f is estimated in each time step, and the values only change slightly in between consecutive steps, then the estimate from the previous time step can be used as an initial guess, and one or a few iterations are usually sufficient for convergence.

Feedback Control Strategies

In order to enforce the constraints specified for the outputs, the components of y had independent scalar feedback controllers in (9.10). One of the following three controllers was used for each component, depending on the type of constraint.

Position control Each output component that had a position reference was governed by a proportional controller:

$$\dot{y}_d^\circ = \dot{y}_d + K(y_d - y), \tag{9.19}$$

where y_d is the reference (desired) output value and K is the proportional gain. The task of this type of controller was mainly to handle errors that were accumulated when integrating the velocity references generated by (9.12), while load disturbances were handled by the low-level joint controllers.

Velocity control Velocity control was particularly easy, since the iTaSC framework used was velocity based. The controller used only included a feedforward term:

$$\dot{y}_d^\circ = \dot{y}_d, \tag{9.20}$$

where $\dot{y}_d$ was the desired velocity.

Impedance control Impedance control was used to make the robot behave as if it were a body with mass M, a viscous damping D and a force F_{ref} acting on it. This was realized by

$$\ddot{y}_d^\circ = \frac{1}{M}\left(F - F_{ref} - D\dot{y}_d^\circ\right), \tag{9.21}$$

where F is the measured force acting on the robot, given in the space of the outputs.

Position-Controlled Motions

Motions that did not require any force feedback could be specified using standard programming languages for industrial robots. For this purpose

the ABB RAPID language was used. It let the programmer specify target poses of the robot, maximum speed and acceleration for different motions, and whether the trajectory generation should be done in joint space or Cartesian space, etc. The RAPID language was easier to use than the above described interface for force-controlled motions and should therefore be used whenever possible.

Execution Platform

A framework for executing iTaSC-based motion control was implemented in Matlab/Simulink, performing the most time-critical and low-level operations, including controller calculations, overload protection, coordinate transformations, kinematics calculations, and gravity compensation. The program was compiled by the Real-Time Workshop toolbox [Real-Time Workshop, 2011] and executed on a real-time Linux PC [Xenomai, 2013], which connected to the robot controllers via the ExtCtrl interface [Blomdell et al., 2010].

The iTaSC-based skills were described by state machines implemented in JGrafchart [Årzén, 2002; *JGrafchart* 2013]. Each state generated a set of outputs specifying the behavior of the motion controller described in the previous paragraph. The outputs from the state machine consisted of the following time-variable parameters:

- kinematic chains to be used;
- active outputs;
- reference values for the outputs;
- controller types (position, velocity or impedance);
- controller parameters.

The conditions for state transitions were typically that a force was detected or that a position was reached.

The main program flow was handled in RAPID, where the main program could execute skills that were implemented either in RAPID (for position-based control) or JGrafchart (for operations that needed force sensing).

9.3 Shield-Can Use Case

A part of a mobile phone assembly was chosen as a use case for robotic assembly. The operation considered was the attachment of a shield can to the PCB (printed circuit board) of the phone, see Fig. 9.2. The shield

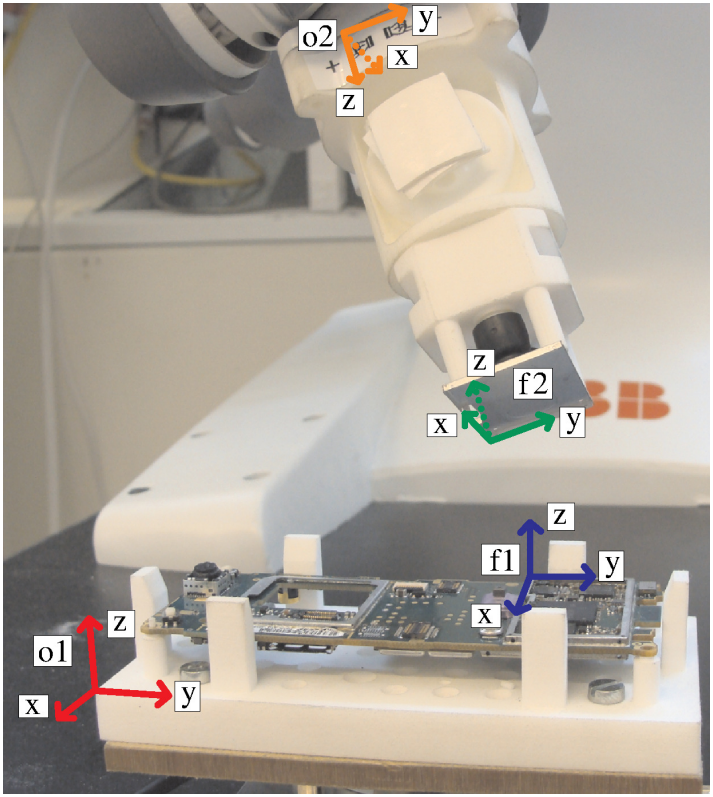


Figure 9.2 Illustration of the frames used in the shield-can assembly task.

can had flexible edges and was attached by pressing it onto a socket on the PCB. The size of the shield can was 38 mm × 23 mm. This section describes how the assembly was performed with a robot.

Tooling

To make it possible to perform the mobile phone assembly, special tooling was produced. A fixture was designed for keeping the PCB in position. A suction tool was used to grasp the shield can, see Fig. 9.3. The maneuverability in contact was good in the vertical direction (the $f2$ z-direction in Fig. 9.2), but worse orthogonal to this direction ($f2$ x- and y-directions in Fig. 9.2), since the shield can might slide. A 6 degrees-of-freedom ATI Mini40 force/torque sensor was mounted beneath the PCB fixture (see Fig. 9.4).

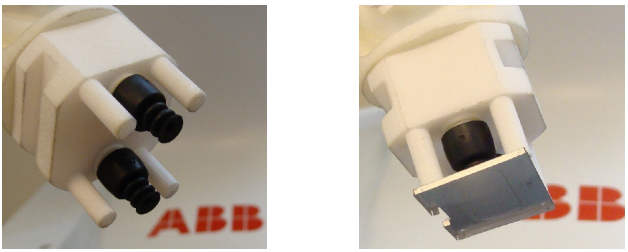


Figure 9.3 The vacuum gripper used in the experiments, without and with the shield can.

Task Modeling

The task was modeled using the iTaSC framework, described in Sec. 9.2. The assembly operation was described using the four frames that are shown in Fig. 9.2 and listed here:

- Object frame $o1$ was attached to the fixture holding the PCB.
- Feature frame $f1$ was attached to a corner of the socket on the PCB.
- Feature frame $f2$ was attached to a corner on the shield can.
- Object frame $o2$ coincided with the flange frame of the robot.

The transformation between frames $f1$ and $f2$ consisted of a series of three translations along the axes of frame $f1$, followed by three reorientations forming XYZ Euler angles.

Uncertainties in the task included the exact location and orientation of the fixture holding the PCB, and also how the shield can had been gripped. The uncertainty of the PCB pose could be modeled by introducing an uncertainty frame $f1'$. This frame represented the modeled position of the socket corner on the PCB, while $f1$ gave the real position. It was assumed that the fixture was mounted such that the PCB was placed in the horizontal plane, but the exact location and orientation in this plane was uncertain. This was modeled by introducing three uncertain translations and one uncertain reorientation angle (around the $f1$ z-axis), see Fig. 9.4. Similarly, in the grasp the orientation around the z-axis and the translations along the x- and y-axes in frame $f2$ were uncertain. The uncertainties were assumed to be a few millimeters and a few degrees respectively.

Assembly Strategy

The assembly strategy was designed such that the uncertainties were resolved in a robust way. A suitable strategy was to put the shield can

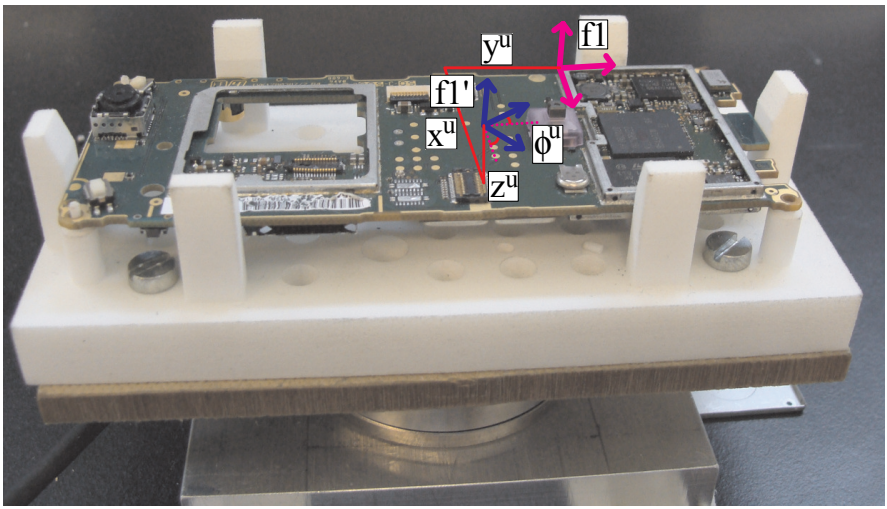


Figure 9.4 Illustration of the uncertainties in the shield-can assembly task. The magnitude of the uncertainty is scaled up to make room for the frames and the labels. A small part of the cylindrical force sensor can be seen underneath the fixture.

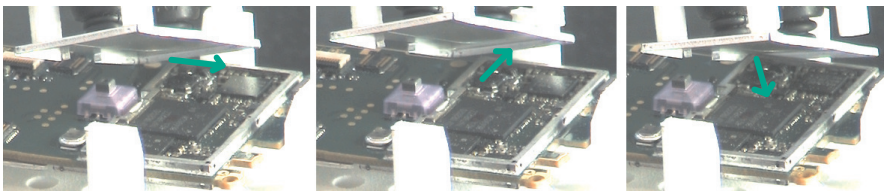


Figure 9.5 Snapshots from the shield-can assembly sequence to illustrate how the corner was found. The arrows indicate in which direction the shield can was in contact. In the leftmost photo the robot was in the end of state 4 and had sensed contact in the y-direction, in the middle photo the robot was in the end of state 5 and had sensed contact in the x-direction, and in the rightmost photo the robot was in the end of state 6 and had found the corner.

in a tilted position, see Fig. 9.2, and then find a corner of the socket by executing a sequence of guarded search motions, i.e., the search motions were stopped once the appropriate contact forces were sensed. Once the corner was found, the shield can was rotated to the correct orientation to press it onto the socket.

A suitable corner to try to find was the one where frame $f1$ is placed, see Fig. 9.2. The area in front of this corner had almost no components that the shield can could get stuck on when sliding over the PCB surface during the assembly. It was further large enough to be possible to find, considering the position uncertainties involved.

A detailed assembly sequence is given below, and snapshots from a few of the steps are shown in Fig. 9.5.

1. Pick up shield can from tray
2. Go to start position
3. Search for contact in negative $f1$ z-direction
4. Search for contact in positive $f1$ y-direction
5. Search for contact in negative $f1$ x-direction
6. Find corner of socket by another search in positive $f1$ y-direction
7. Make a rotational search around the $f2$ x-axis and the $f2$ y-axis
8. Press shield can into position
9. Release shield can and move away from PCB

9.4 Emergency-Stop-Button Use Case

The assembly of an emergency stop button was used as a second use case for robotic assembly. An assembly graph for the scenario is displayed in Fig. 9.6.

In the *top subassembly* in the upper right corner of Fig. 9.6 the red button should be inserted into the hole of the yellow box, and then be attached by screwing a nut onto the black part of the button. In the *bottom subassembly*, shown in the upper left corner of Fig. 9.6, the electrical switch should be attached to the gray box by a snap-fit operation. Finally the top subassembly should be put on top of the bottom subassembly to finish the assembly task. The dimensions of the bottom surface of the box was 101 mm $\times$ 74 mm.

Workcell

A photo of the workcell that was designed for the assembly scenario is displayed in Fig. 9.7. The robot was equipped with three different gripping tools to accomplish the assembly task. All tools were mounted on tool exchangers, and a tool stand was used to enable tool exchanging during the assembly procedure. The different components for the assembly were placed in different trays in the workcell. The metal plate in the bottom of Fig. 9.7 was mounted on a force sensor and could be used as a fixture for a gray or a yellow box during different parts of the assembly.

Force measurements could be acquired either from the force sensor under the fixture or from the internal sensing of the robot, using the methods described in Chapter 10.

Set of Skills

The assembly task was divided into the following set of skills, described in more detail on pages 155–164.

- Pick yellow case
- Put yellow case on fixture
- Pick red button
- Insert red button into yellow case
- Pick nut
- Lift red button and yellow case and turn them around
- Screw nut
- Align yellow case against fixture
- Check whether yellow case should be turned 180 degrees
- Put yellow case in intermediate storage
- Change tool
- Pick gray box
- Put gray box on fixture
- Pick switch
- Do snap fit of switch
- Put switch and gray box on table

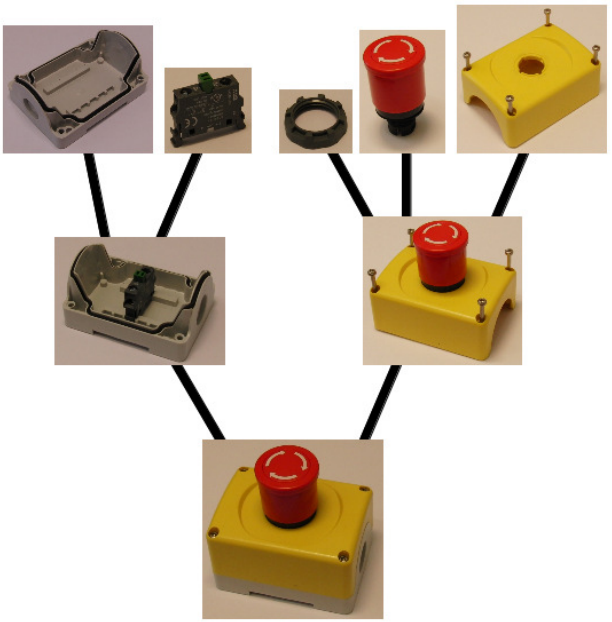


Figure 9.6 Assembly graph for the emergency stop button assembly scenario.

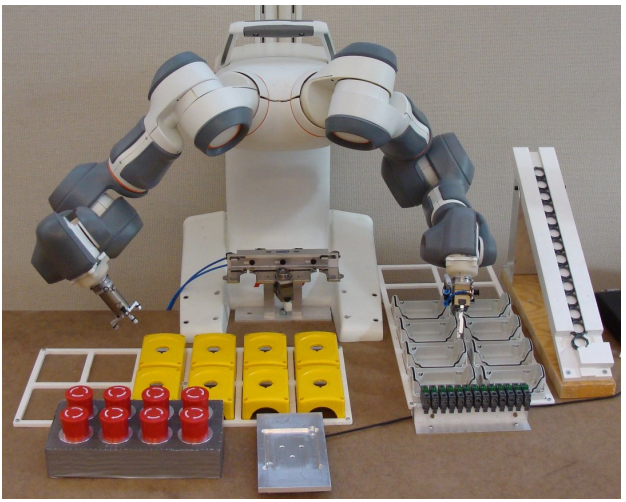


Figure 9.7 The workcell for the emergency stop button assembly scenario.

- Pick yellow case from intermediate storage
- Put yellow case on top of gray box and slide them to the side
- Change tool

Different tools were needed when assembling the top and bottom subassemblies. In order to avoid excessive tool exchanges, several top subassemblies could be assembled first, and put in an intermediate storage tray. After that, one arm changed tool to assemble the bottom subassembly. When a bottom subassembly was finished, a top subassembly was picked from the intermediate storage and put on top of the bottom part to finish the assembly operation.

Position-Controlled Skills

All picking skills could be performed using pure position control, since all parts were placed in well-defined trays, and the design of the grippers allowed some tolerance on the gripping position. The tool-exchange skill was another position-controlled skill, which was performed with a fixed tool stand. Putting the gray box with the switch on the table and putting the yellow box on top of it could also be done with position control, since the contact surfaces were slightly slanted, reducing the position accuracy needed for this operation.

Picking a nut was an operation where significant position uncertainty was present, but it was resolved without using any force sensing. The nuts were picked from a slide, which can be seen in the right part of Fig. 9.7. The bottom part of the slide was designed such that the nut to pick should end up in a pre-determined position, but when a nut had been picked and the rest of the nuts slid down, it quite frequently happened that the next nut to pick did not end up in the desired position. Examples of different nut positions are displayed in Fig. 9.8. The position could be corrected by always pushing the nut, first from the right side and then from the left side, before the nut was picked, even when the position was correct in

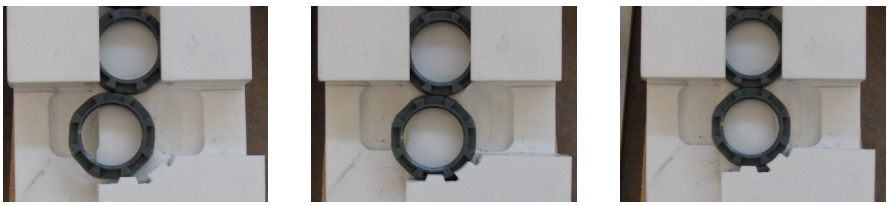


Figure 9.8 Close-ups of the nut slide. *Left:* Nut too far to the left. *Middle:* Nut in the correct position. *Right:* Nut too far to the right.

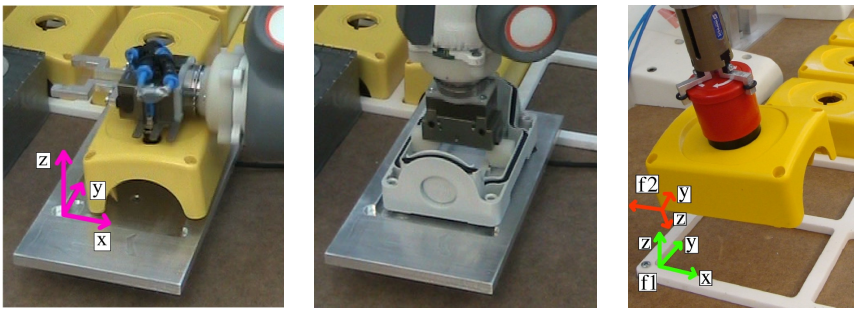
the first place. In the absence of appropriate feedback signals to measure the position of the nut, this type of robust picking skill could be used to prevent errors.

Force-Controlled Skills

The skills described on pages 156–164 required force sensing to be performed robustly. Most of them contained *guarded search motions*, which searched for objects by moving along a path until a contact force in the direction of motion was detected. Once the contact was found, the position where it took place could be used as the base for a position reference in the subsequent motions, or the contact force could be maintained using force control. The force sensing could also be used for supervision, e.g., detecting if contact forces became too large and risked damaging equipment.

Putting boxes on the force sensor or intermediate storage Putting a yellow or gray box on the force sensor was performed by three search motions. First, the box was moved to an initial position close to the force sensor, using position control. The initial position was slightly further in the positive directions of the x-, y-, and z-axes than the final position, as shown in Fig. 9.9. The box was then moved in the negative z-direction until it made contact with the force sensor and a force was detected. While using force control to maintain the contact force in the z-direction, a search motion was then performed in the negative x-direction. Finally, a search motion in the negative y-direction was performed until the box slid down into the slot and a contact force in the y-direction was detected.

Putting the top subassembly in the intermediate storage was performed in a similar manner, but it started in a tilted pose, as shown in Fig. 9.9(c), so that the corner at the origin of frame $f2$ was the first to make contact with the table. Once the corner at the origin of frame



(a) Yellow box on force sensor. (b) Gray box on force sensor. (c) Intermediate storage.

Figure 9.9 Putting boxes on the force sensor or in the intermediate storage tray.

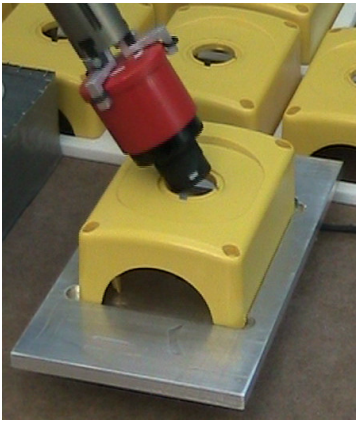


Figure 9.10 Putting the red button on the yellow box, a peg-in-hole operation.

$f1$ had been found by performing three guarded search motions, the subassembly was rotated so the $f2$ z-axis pointed straight downward and the subassembly stood securely in the tray.

Peg-in-hole insertion of red button into yellow box Putting the red button in the yellow box was a peg-in-hole operation. In the approach phase the button was tilted as in Fig. 9.10. The tilted orientation gave the button a more pointed contact surface, and hence less position accuracy was needed to hit the hole in the yellow box.

Once contact was made, the button was pressed downward and the forces in the radial directions of the hole were controlled to zero in order to center the button in the hole. The button was then tilted until the center axis was in the vertical direction and the button slid down in the hole due to a force reference pressing the button downward.

Screwing The screwing skill was the most complex skill. It was performed to fasten the button to the yellow case by screwing the nut. One robot arm held the button with the case on top, and the other arm held the nut and performed the screwing. The initial configuration before the screwing was started is displayed in Fig. 9.11, and the frames used for modeling the relative motion between the arms are illustrated in Fig. 9.12.

Two kinematic chains were used for the nut-screw scenario. Chain a was used to specify the relative motion between the arms, and chain b was used to specify the motion of the arm holding the button with the case on top. For both kinematic chains, the feature coordinates described the transformation between frames $f1$ and $f2$; first three translations along the axes of $f1$ and then three Euler angles to describe the reorientation.

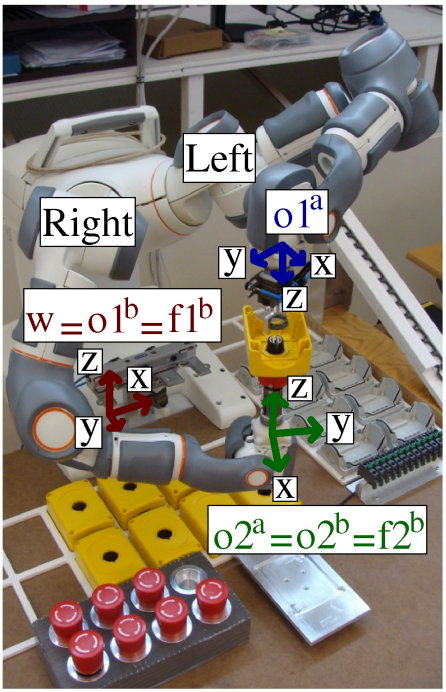


Figure 9.11 Illustration of the two kinematic chains used in the nut-screw assembly skill. Chain a was used to specify the motion of the left arm w.r.t. the right arm. Chain b was used to specify the motion of the right arm w.r.t. the world frame.

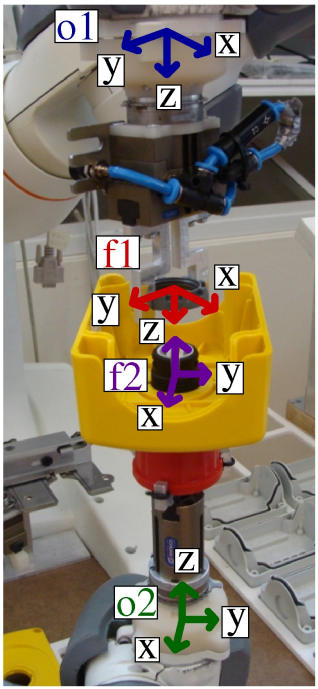


Figure 9.12 A detailed view of kinematic chain a in the nut-screw assembly skill. The superscript of the frames has been omitted in this figure.

As an example, increasing the z-coordinate in chain a resulted in the arms moving apart.

The nominal assembly strategy was as follows

1. Initial positioning
2. Put nut on the button by a search in $f1^a$ z-direction
3. Re-grip nut
4. Screw until nut is tightened
5. Release nut and move away

Because of various types of uncertainties this assembly strategy was very unlikely to succeed. To make the execution more robust, both proac-

tive and reactive schemes were included in the assembly strategy. A sketch of the actual state machine coordinating the assembly skill is displayed in Fig. 9.13.

To start with, the red button with the yellow case on it was placed in a vertical position, as shown in Fig. 9.12. The other arm, with the nut in the gripper, approached the button and tried to put the nut on it. This was made as a guarded search motion in $f1^a$ z-direction. If the distance between the arms was small enough when a contact force was detected, the nut was put on the button, and this scenario corresponded to the nominal strategy. If the distance between the arms was to large when contact was made, the hole had been missed and the nut got stuck on top of the button. The recovery strategy was to move the robot back, slightly modify the position in the $f1^a$ x- and y-directions, and try again. This continued with a search pattern in the x- and y-positions until the nut was successfully put on the button.

The gripper for the nut could grasp the nut in two ways, as shown in Fig. 9.14. The first kind of grip gave a rigid grip of the nut and was used for putting the nut on the button. It could, however, not be used for screwing, since it would make the gripper collide with the yellow box. The nut therefore had to be released and re-gripped, and this might cause a movement of the nut. To be robust against this type of error, a proactive strategy was applied, namely to push on the nut to make sure it was in contact with the thread on the button. The next step was to go down to the nut, grip it, and start screwing. The robot could screw one revolution before it had to release the nut and rotate back to the original orientation again.

It was known that it took 2.5 to 3.5 revolutions to tighten the nut, depending on the orientation of the nut when the screwing was started. Detecting that the nut was tightened could not be done by means of the estimated torque around the screwing axis, due to too large disturbances in the torque measurements. What usually happened when the nut was tightened, was that one of the two fingers lost its grip of the nut, so the nut was pushed to the side instead of rotated. This resulted in a large side force (a force in the $x-y$ plane) that could be detected. If this happened during the third or the fourth revolution of screwing, it was assumed that the reason was that the nut was tightened. Sometimes the gripper did not lose the grip of the nut when the nut was tightened, and the red button slid in its grip instead during the rest of the revolution. This was what was assumed to have happened if no large side forces had been detected after four revolutions.

During screwing, there was a risk that the nut was gripped in a non-centric way, which could cause large side forces although the nut was not yet tightened. This was particularly likely to happen in the beginning

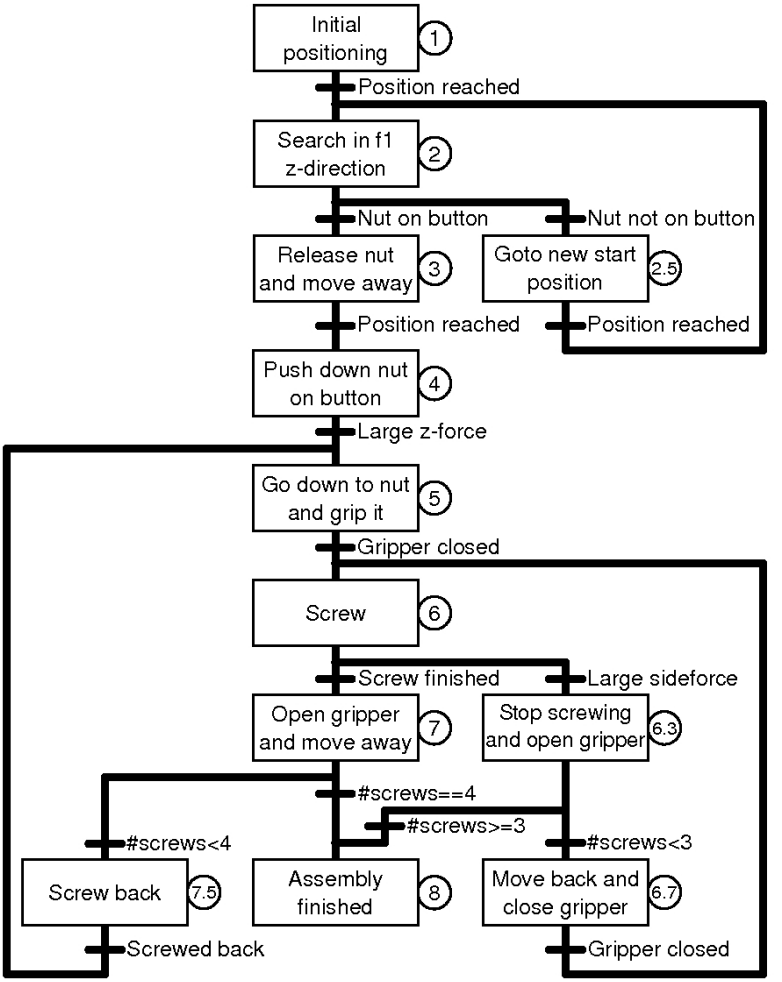


Figure 9.13 The state machine used for modeling the assembly strategy for the nut-screwing skill.

of the screwing, before the nut was properly on the thread. If a large side force was detected during the first two revolutions, the screwing was stopped. The nut was then released, the robot arms are moved slightly apart, the nut was gripped again, and the screwing continued. This set of actions was another example of an automatic error recovery strategy, which led to more robust execution.

(a) Rigid grip used for putting the nut on the button.

(b) 'Finger tip' grip used for screwing the nut.

Figure 9.14 Different grips of the nut.

Snap fit The switch was attached to the gray box by a snap-fit operation. The feature coordinates used to describe the pose of the switch were (x, y, z), parameterizing three translations, followed by a reorientation described by (φ, θ, ψ), forming Euler ZYX angles. The assembly strategy used could handle a position uncertainty of a few millimeters. Figure 9.15 shows snapshots from the assembly sequence with step numbers given by

1. Move to start position.
2. Search for contact in negative z-direction.
3. Start force control in the z-direction and search for contact in the negative y-direction.
4. Start force control in the y-direction and search for contact in the negative x-direction.
5. Start force control in the x-direction and search for contact in the negative ψ-direction.
6. Start torque control in ψ-direction and search for contact in the negative φ-direction as the switch slips down into its slot.
7. Start torque control in the φ-direction and search for contact in the negative ψ-direction.
8. Lift the assembly in the positive z-direction.

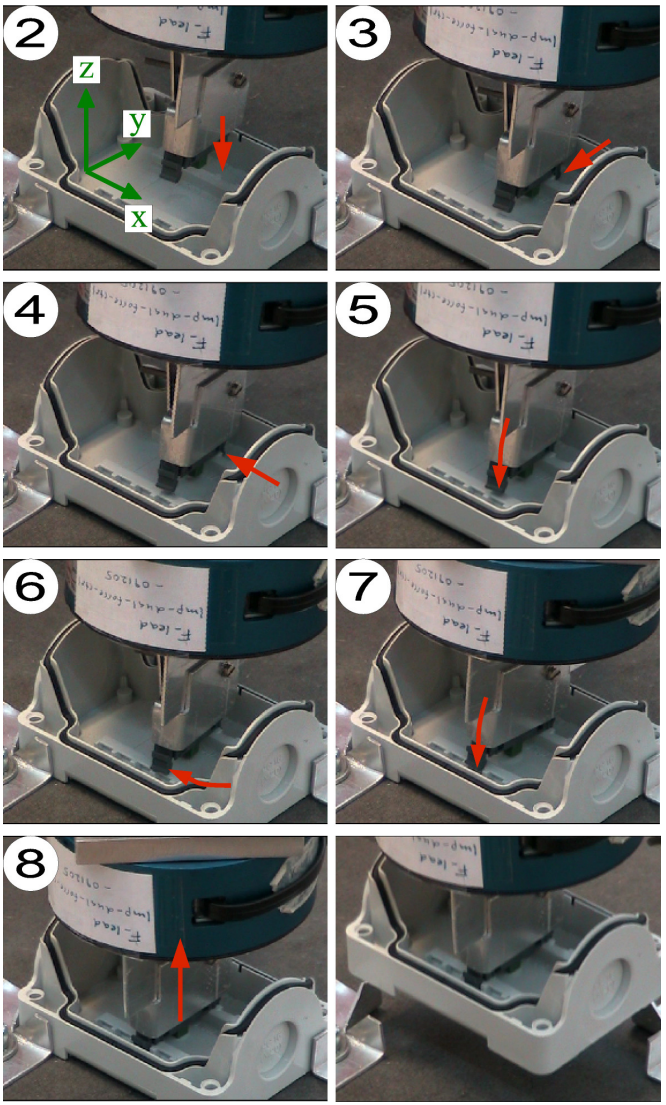


Figure 9.15 Attaching the switch to the gray box by a snap-fit operation. The red arrows show the direction of motion in the different steps.

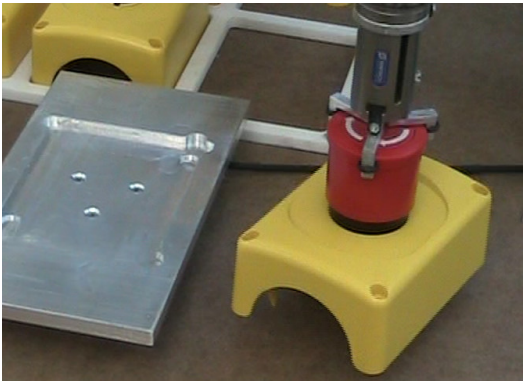


Figure 9.16 Example of initial position for the align box skill. The orientation of the yellow case in the gripper was unknown after screwing due to sliding in the gripper.

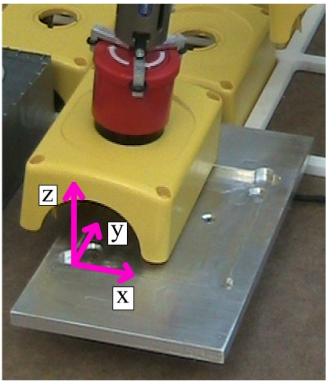


Figure 9.17 Checking the position of the small peg on the yellow box.

Aligning the box after screwing The orientation of the yellow case was unknown after the screwing, as the red button might slide in its gripper. An aligning phase against the fixture on the force sensor was therefore performed. A simple strategy would be to start from the position displayed in Fig. 9.16 and then perform a search motion towards the fixture until contact was established. The box could then be given its desired orientation by pushing it against the fixture and controlling the torque around the symmetry axis of the button to zero. It was, however, difficult to make the torque controller fast without running into problems with instability.

To make the procedure faster, the following strategy was applied, exploiting that the distance between the button and the fixture was the smallest when box was aligned with the fixture.

1. Go to start position (beside the fixture as in Fig. 9.16).
2. Search for contact against fixture.
3. If distance to fixture is not small enough, move away from the fixture, rotate a given angle, and go back to step 2.
4. Rotate to make contact with opposite corner of the same long edge of the case (rotation direction determined from measured torque).
5. Rotate to the angle half way between the angles found in steps 3 and 4.

6. Search for contact against sensor.

7. Control torque to zero.

The skill had two extra phases compared to the simple strategy. The first phase tried out a discrete set of angles (steps 2 and 3) to quickly get to an orientation that was approximately correct, similar to the one in Fig. 9.16. To illustrate the second phase (steps 4 and 5), assume that the box had the same orientation as in Fig. 9.16. The box was then rotated counterclockwise in step 4 until the corner that was behind the red button in the figure made contact with the fixture. Step 5 could then compute an orientation very close to the correct one, so the torque control in step 7 could converge quickly.

Check orientation of yellow case It may look like you could turn the yellow case 180 degrees around the center axis of the hole in the case, and it would still fit on top of the gray box. This rotation was, however, not possible, due to a small peg in the yellow case and a corresponding slot in the gray box, illustrated in Fig. 9.18.

When the assembly was performed, it was assumed that the orientation of the gray box was known in its tray. After the screwing and aligning of the yellow case, it was not known whether the case was in its desired orientation or turned 180 degrees.

To determine the orientation of the yellow case, it was put in contact with the fixture on the force sensor, as shown in Fig. 9.17. The box was then moved in the positive y-direction, and if a force in the y-direction was detected by the force sensor, the peg was in the corner with the largest x- and y-coordinates. If the box could be moved a given distance without

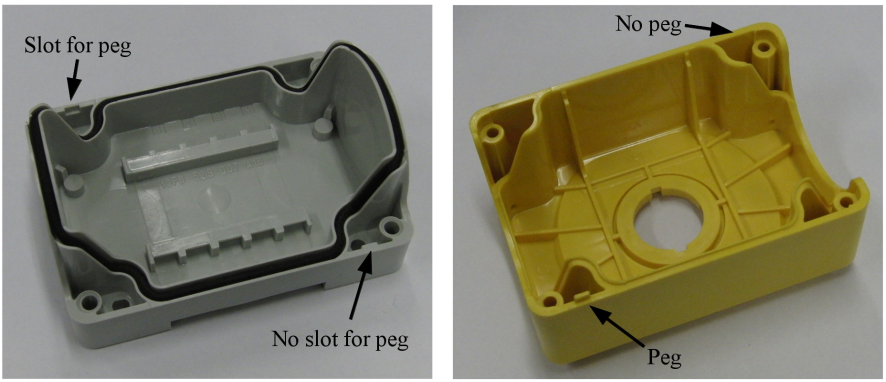


Figure 9.18 Image showing the small peg on the yellow case and the matching slot on the gray box.

any detected force in the y-direction, the peg was in the corner with the smallest x- and y-coordinates.

9.5 Experimental Results

Robots

All assembly operations described in Secs. 9.3 and 9.4 were performed using the ABB FRIDA robot [Kock et al., 2011], see Fig. 9.7. It is a dual-arm manipulator developed for automation of assembly operations. It has low mass, weak motors, and padded surfaces to be able to work safely next to humans. Each of the two arms is redundant with 7 degrees of freedom. The snap-fit operation, attaching the switch to the gray box, was also performed with an ABB IRB 140 robot [ABB, 2013], equipped with a wrist-mounted JR3 100M40A force/torque sensor [JR3, 2013]. The IRB 140 is a conventional 6-axis industrial robot with a maximum payload of 6 kg, reach of 810 mm and position repeatability of ± 0.03 mm. An image of the robot can be seen in Fig. 2.2. The robots were controlled with the ABB IRC5 robot control system. This system was extended with an open control system [Blomdell et al., 2005; Blomdell et al., 2010], which made it possible to modify the references for the low-level joint control loops.

An ATI Mini40 6-DOF force/torque sensor was mounted under the PCB for the shield-can assembly and under the fixture for the emergency-stop-button assembly. Contact forces could also be estimated by using internal sensing of the robot, namely the joint control errors or the motor torques of the robot, as described in Chapter 10. The force estimation based on internal sensing was less accurate than the measurements from the force sensor, but it was necessary for operations that could not be performed on the force sensor, e.g., screwing the nut and putting the top subassembly into the intermediate storage.

Shield Can

The implementation of the shield-can assembly achieved an assembly time of approximately 4 s, slightly longer than it would take a human to do it. A video of the assembly can be accessed through [Stolt et al., 2012]. In an experiment where the assembly operation was performed 30 times, the attempts succeeded 25 times. To make the task extra challenging, the shield can was gripped in different ways by moving it up to 3 mm from its nominal pick-up position. Had the shield can been displaced farther, it would have ended up out the supporting pins of the gripper, see Fig. 9.3, and could not have been gripped properly. When the assembly task failed it was usually due to one of the following reasons:

- In step 5 of the assembly sequence (p. 152), the shield can was *pushed* along the surface of the PCB, instead of *pulled* as in the other sliding operations. Because of this, the shield can could get stuck and cause large x-forces before before reaching its target contact surface.

- When the shield can was pushed against the corner of the socket in step 6, the compliance of the grip let the shield can rotate around the $f2$ z-axis so it aligned itself with the socket. Sometimes this alignment failed, so the socket could not be pressed onto the socket in step 8.

- Sometimes the shield can slipped over the corner when it searched for a corner of the socket in step 6.

Emergency Stop Button

The snap-fit attachment of the switch to the gray box was performed both with an IRB 140 and a FRIDA. The IRB 140 performed the operation in 9 s, while FRIDA did it in 4 s. Since IRB 140 is much stiffer and heavier than FRIDA, contact forces increased much faster for IRB 140 than for FRIDA when they made contact at the same speed. Hence, the search motions had to be made much slower for IRB 140 than for FRIDA to have time to react to contact forces before they became too large.

The entire assembly sequence for the emergency stop button was implemented with FRIDA. A video of the assembly is given in [Stolt et al., 2013]. The total time for one cycle was 59 s. The measured time for the different skills are given in Table 9.1. Some of the skills could be performed in parallel, such as the picking of the red button and the picking of the yellow box, while some skills had to wait for other skills to finish using the shared resources, i.e., the fixture together with the force sensor.

The cycle time for a human performing the assembly sequence was about 10–15 s, if the human was allowed to perform the assembly without restrictions. If the human had to perform the assembly using the fixture, in the same way as the robot, then the cycle time became 17 s. The measured time for each particular skill is given in Table 9.2. The human obviously did not need to change tool, and could also skip the alignment skill. Further, the human also waited with checking the 180 degree orientation of the yellow box until finalizing the assembly and putting it on top of the gray box.

Skill	Time	Right arm	Left arm
Pick yellow box	3 [s]	-	X
Put yellow box in fixture	2 [s]	-	X
Pick red button	3 [s]	X	-
Insert red button in yellow box	5 [s]	X	-
Pick nut	4 [s]	-	X
Turn yellow box and red button	5 [s]	X	X
Put nut on thread	6 [s]	X	X
Screw nut	13 [s]	X	X
Move from screwing to alignment	4 [s]	X	-
Align yellow box	2 [s]	X	-
Check orientation	3 [s]	X	-
Put in intermediate storage	6 [s]	X	-
Change tool	6 [s]	-	X
Pick gray box	3 [s]	-	X
Put gray box in fixture	2 [s]	-	X
Pick switch	2 [s]	-	X
Do snap fit	4 [s]	-	X
Put gray box and switch on table	1 [s]	-	X
Pick from intermediate storage	2 [s]	X	-
Put yellow box on top of gray box	3 [s]	X	-
Change tool	5 [s]	-	X

Table 9.1 Time taken for the skills in the assembly scenario with the robot implementation. The table also shows which of the arms was performing which skills.

9.6 Discussion and Lessons Learned

Based on the experiments performed, the cycle time of the robot implementation was approximately 3.5 times slower than that of a skilled human assembly worker. The robot implementation could be speeded up a little bit, by for instance optimizing trajectories for free space motions and minimizing the time the arms had to wait for each other during the assembly sequence. It is, however, not reasonable to believe that the cycle time can be made as fast as the human with the current workcell, i.e., with the only external sensor being the table-mounted force sensor. The human has several superior advantages. One is the vision feedback coming from the eyes, which makes it possible to start all assembly operations much closer to the contact configurations. For many assembly operations the human used force/tactile feedback, with a strategy similar to that of the robot, but the human force control capabilities are much better than those of the robot for the purposes of these assembly operations.

Skill	Time	Note
Pick yellow box	1 [s]	
Put yellow box in fixture	1 [s]	
Pick red button	1 [s]	
Insert red button in yellow box	1 [s]	
Pick nut	2 [s]	
Turn yellow box and red button	2 [s]	
Put nut on thread	0 [s]	Done while turning
Screw nut	4 [s]	
Move from screwing to alignment	0 [s]	Skipped by human
Align yellow box	0 [s]	Skipped by human
Check orientation	1 [s]	
Put in intermediate storage	2 [s]	
Change tool	0 [s]	Not necessary
Pick gray box	1 [s]	
Put gray box in fixture	1 [s]	
Pick switch	1 [s]	
Do snap fit	1 [s]	
Put gray box and switch on table	1 [s]	
Pick from intermediate storage	1 [s]	
Put yellow box on top of gray box	1 [s]	
Change tool	0 [s]	Not necessary

Table 9.2 Time taken for the skills in the assembly scenario with the human.

The robot performed the screwing in 13 s, and the human in 4 s. The human hand is really much more suited for this kind of task than the gripper used for the robot, as the human can use a finger to spin the nut to efficiently fasten it. If the robot would not have needed to screw back after each revolution, because of limits of the robot joints, then the time needed for the screwing would be more than halved, and thus comparable to that of the human. It could for instance also be possible for the robot to use a specialized screwing tool in the work space. Then it could actually be possible to do it faster than the human.

An advantage with a robot is that it can work 24 hours per day, and 7 days per week, and it will in this time have enough time to complete about 10 000 stop buttons (with a cycle time of 59 s). This can be compared to a human that works 8 hours per day, and 5 days per week, who will be able to complete about 8 500 stop buttons. In this sense, the current robot implementation has a cycle time comparable to that of a human assembly worker.

The analysis so far has been made with the assumption that nothing

goes wrong. Some of the skills were extended, so that they could detect and recover from common errors, but there were still many errors that were not handled by the system. The success rate was approximately 75 % when the system was properly tuned, and when the assembly failed it might take a couple of minutes for a human to restore the workcell. Producing 10 000 stop buttons per week with the current implementation is therefore unrealistic. The most realistic path towards the goal of reaching the same productivity as a human is probably to make the robot capable of detecting and recovering from more errors, rather than making sure no errors occur.

The brief description of the emergency stop button on page 152 would probably be sufficient for most humans to perform the assembly. It is even likely that many people could perform the assembly if they only were provided with the parts in the top line of Fig. 9.6, without any instruction. In contrast, the simplified description of how the robot was programmed takes up pages 152–165. This hints at the vast amount of prior knowledge that humans use to perform such operations. Replicating similar capabilities in robotic systems, being able to perform assembly with minimal task-specific instructions, provides a huge challenge for the future.

The picking operations were performed using position control, since the available sensing was not sufficient to perform force-controlled picking. Many of the parts would tilt out of their positions if they were pushed in their trays, and the force estimation based on internal sensing could not detect sufficiently small forces to use force control for picking. On a few occasions during the work on the stop button, the positions of the assembly task had to be re-tuned, since the robot was exchanged by another robot or the workcell was moved to a new table. Re-tuning all positions of the picking skills and tool switching with high precision was tedious work. The force-controlled skills required much less precision on the starting positions, and starting positions that worked were much easier to find, if they had to be modified at all. These events demonstrated the power of force control w.r.t. position disturbances. On the other hand, the force-controlled skills required more work the first time they were programmed, since they needed tuning of force thresholds and force-feedback parameters.

The snap-fit operation performed to attach the switch to the gray box was implemented for both IRB 140 and FRIDA, with the conclusion that FRIDA was much more suited for small-parts assembly. A very common operation in the force-controlled assembly operations was to perform a search for contact, meaning that the robot was moving along a path until a contact force was detected. The contact event could be seen as a collision, and if the robot did not stop quickly the contact force would continue increasing, possibly to levels that might damage equipment. In comparison

to FRIDA, IRB 140 was stiff and heavy, meaning that the contact force increased much faster when contact was made. Due to these properties, FRIDA could make much faster search motions without risking too large contact forces, thus finishing the assembly operations faster.

In a few skills where we expected force control to be useful, it turned out that other strategies were more useful. One example is in the assembly sequence for the shield can, see page 152. Once contact had been made with the PCB in step 3, the contact in the z-direction during the subsequent sliding motions could be maintained using force control, but the friction disturbances from sliding on the PCB made it difficult to both move fast and control the z-force. In this example the orientation of the PCB was well known and there was some compliance in the grip of the shield can, so steps 4–6 could be performed using position control in the z-direction, and the motions could be made much faster than when force control was used in the z-direction.

The alignment skill on page 163 is another example of where the use force/torque control was not the best solution. The first approach implemented was to push the box against the force sensor and control the torque around the symmetry axis of the button to zero. It was, however, difficult to design a fast well-damped torque controller. A strategy that turned out to be much faster was to search for contact with the force sensor for a fixed set of rotations of the box. This strategy quickly found an approximately correct angle, and with such a good initial angle, finding a more correct alignment angle, using torque control, did not take very long.

The iTaSC framework was chosen as the base for specification of force-controlled motions due to its generality. It served its purpose well, but in practice most motions were described by a single kinematic chain with three translations along the basis axes of frame $f1$ of the specific motion, followed by three rotations describing the orientation in an Euler-angle fashion. These parameterizations did not require the full power of the iTaSC framework, but it was good to have the functionality in place if the need should turn up.

Using Euler angles for describing orientation can be problematic due to the existence of representation singularities. They were still extensively used for task specification in the implementations described in this chapter, since they are intuitive to use. They also provide a convenient way to specify operations where torque control is used for one or two directions, while position control is used for the remaining rotational degrees of freedom, as in steps 6 and 7 of the snap-fit operation described on page 161.

9.7 Conclusions

The work presented in this chapter demonstrated how robots can be used to assemble objects designed for human assembly. The assembly speed was, however, significantly lower than for a human worker and the failure rate was far too high to be used in the industry. Much work on increasing the speed and robustness still lies ahead of us before such a system can be used for production.

10

Robotic Force Estimation without any Force Sensor

10.1 Introduction

The traditional way of programming industrial robots is to use position control and follow pre-defined trajectories, using the joint position sensors. Modern robot controllers implement this strategy well and perform motions both fast and with high accuracy. In tasks where the robot has to physically interact with the environment, however, this control strategy is less advantageous. The accuracy of the robot and the location and geometry of everything in the work space have to be known with high precision, and this is usually hard to achieve. A remedy to this problem is to introduce additional sensing, e.g., a force sensor that gives the robot capabilities to handle position uncertainties by sensing the contact forces.

Different sensor setups are possible, e.g., using a force/torque sensor mounted on the wrist of the robot or under a fixture, or having one torque sensor in each joint as in the DLR light-weight robot [Albu-Schäffer et al., 2007]. The main drawbacks with using force sensors are that they may be very expensive and add weight to the system. A 6-axis force/torque sensor for industrial use costs roughly $10 000, while the price of a 3-axis force sensor is approximately $1 000. This should be compared to a typical small industrial robot, which costs $20 000 – $30 000. The DLR light-weight robot with integrated joint torque sensors costs approximately $120 000.

An alternative to using force sensors is to estimate the external forces applied to the robot based on sensing already available in the robot. Usually this includes position sensors in the joints, and torques exerted by the motors. A significant problem with using motor torques is that they are affected by disturbances originating from friction, the mechanical transmission between the motor and the robot link, etc. In contrast, the torque

sensors of the DLR light-weight arm are mounted on the link sides of the joints and are, hence, not affected by these particular disturbances.

Force estimation has been considered many times before in the literature. A common approach is to use model-based disturbance observers. A dynamical model of the robot is used and forces are estimated by an observer based on the deviations from the model. Apart from a model of the robot, these methods need a model for interaction with the environment. Examples of previous work using this technique can be found in [Ohishi et al., 1992; Eom et al., 1998; Alcocer et al., 2003].

Force estimation functionality is also available in a commercial industrial robot system by Toshiba [Toshiba, 2012]. Only a brief description of the principles used is available. The system is, however, designed for and reported to work well for force magnitudes of 25–75 N, and would therefore not be suitable for use in tasks where it is important to react already for forces of a few Newtons, which is considered in this chapter.

In previous work, e.g., [Murakami et al., 1993; Ohishi, 1993; Rocco et al., 1997; Simpson et al., 2002], it was assumed that the joints were always moving, and no attention was given to the large uncertainties in the friction torques at velocities close to zero. It has also been common to only consider experiments with three or fewer joints.

In [Popovic and Goldenberger, 1998; Simpson et al., 2002] the performance of force estimation was improved by modeling the friction carefully and considering position-dependent torque variations, and [Olsson et al., 1998; Du and Nair, 1999] included modeling of low-velocity friction phenomena. These models, however, require knowledge of many parameters that are challenging to identify and prone to change, due to temperature and wear etc. Making good estimates of the low-velocity friction torques may also require velocity measurements of a high accuracy that is not available on standard robots.

The following two sections will present two new methods for performing robotic force estimation without any force sensor.

10.2 Robotic Force Estimation Using Joint Position-Control Errors

Method

This section describes an approach for estimating forces that can be used when each joint on the lowest level has an individual position controller, which is a standard solution in industrial robots. By disabling the integral action in the joint controllers, they will act as virtual springs, and the deviation of each joint angle from its reference will correspond to a joint torque, as illustrated in Fig. 10.1. Due to friction and gravity, the

Figure 10.1 Illustration of how the joint torques are estimated. The controlled joint acts as a virtual spring, and when a force is applied to it, the deflection is used to estimate the joint torque.

joint position errors may become large if the integral action is removed completely, leading to poor performance in the robot positioning and bias in the force estimate. One remedy to this problem is to use a small integral part, which allows force transients to be detected, but over time the position errors will be removed. Estimation of forces based on joint errors, using small integral action, results in a high-pass filtered version of the forces.

The previous paragraph proposed a method for estimating the joint torques, but usually the contact forces at the tool are more useful. The joint torques, τ, and the end-effector forces, F, are related by

$$\tau = J(q)^T F + e, \tag{10.1}$$

where $J(q)$ is the robot Jacobian, q is the robot joint joint coordinates, and e are disturbance joint torques with the assumption $\mathrm{E}[e] = 0$ and $\mathrm{E}[ee^T] = R_e$. The minimum variance estimate of the force is then given by

$$\hat{F} = (JR_e^{-1}J^T)^{-1}JR_e^{-1}\tau, \tag{10.2}$$

but if the disturbances are large, the estimate may be of very poor quality. By adopting a Bayesian approach and using prior knowledge about the particular assembly operation, it may be possible to improve the force estimates. Assume that the prior knowledge about F can be described by $\mathrm{E}[F] = \bar{F}$ and $\mathrm{E}[(F - \bar{F})\,(F - \bar{F})^T] = R_F$, and that the distribution of τ conditioned on F is given by (10.1), then the minimum-variance unbiased estimate of F is

$$\hat{F} = (JR_e^{-1}J^T + R_F^{-1})^{-1}(JR_e^{-1}\tau + R_F^{-1}\bar{F}) \tag{10.3}$$

For example, it may be known that the contact torques on the end effector may be very small during an assembly operation. By reflecting this

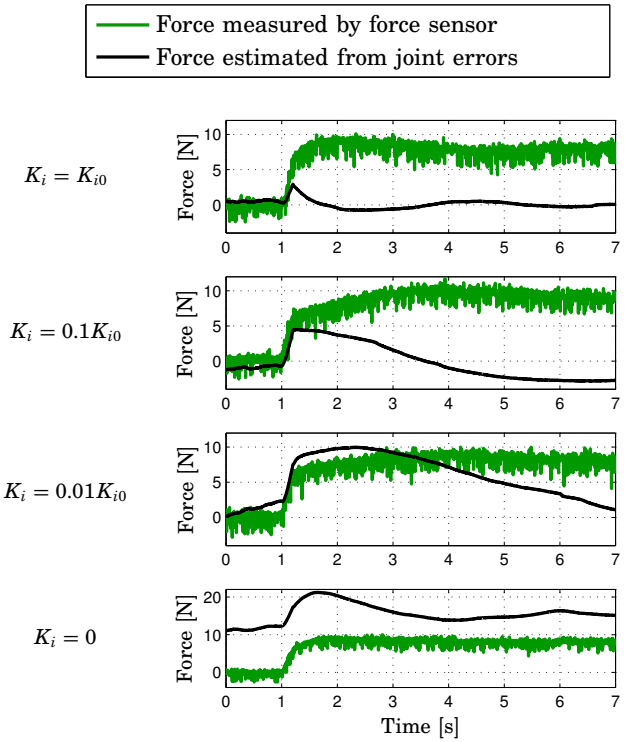


Figure 10.2 Estimated force and measured force in one direction for different values of the integral gain, K_i, in the joint controllers. The nominal integral gain is K_{i0}.

knowledge in the choice of the $\bar{F}$ and R_F used for the calculations, the estimates of the contact forces can be improved.

Calibration

The force estimation is affected by how the detuning of the joint controllers has been performed. If the integral part is removed completely, gravity and friction cause offsets in the positions, which in turn cause offsets in the estimated forces. Keeping the integral part, however, makes it impossible to estimate constant forces, since the integral action eliminates stationary errors. The position-control errors will then resemble the contact forces passed through a high-pass filter, allowing detection of transients. The behavior for different detunings of the integral action is shown in Fig. 10.2, where the force sensor was used to find contact in one direction and control the contact force to a constant value. It can

be seen in the diagrams that a high integral gain gives a transient with a short duration, which may be hard to detect. Removing the integral action completely, however, introduces a bias in the estimate. The final controller detuning chosen for use in the assembly task was with the integral gain $K_i = 0.03K_{i0}$, where K_{i0} is the nominal integral gain in the joint controllers.

A large disturbance acting on the force estimates is friction in the joints. Experiments were performed to estimate the friction magnitude in each joint, which mostly consisted of Coulomb friction. These values were used to choose the diagonal elements of R_e, i.e., the variance of the disturbance torques in (10.1). The effect of gravity was assumed to vary slowly, such that the integral part in the joint controllers could compensate for it. Plugging the estimated friction disturbances into (10.2), the force estimation errors were predicted to be in the order of magnitude of 1 N, which agreed with the errors measured during the execution of the assembly task, see Fig. 10.3.

To determine the spring constants of the joints, forces or torques were applied to the tool of the robot, and the amplitude of the resulting joint error transients were recorded. Doing this for three different arm configurations, it was possible to determine the stiffness of all joints.

Experimental Results

The shield-can assembly scenario, described in Sec. 9.3, was used to evaluate the force estimation. The assembly was performed by an ABB FRIDA robot. An ATI Mini40 force/torque sensor was mounted under the PCB fixture to provide ground-truth data, but it was not used for control during the assembly experiments.

The major part of the assembly was performed with only a point contact, coinciding with the origin of frame $f2$, see Fig. 9.2. Hence, the contact torques around this point should be zero. Modeling errors of course contributed to some torques, but they should be small. This insight could be used as prior information, i.e., it could be used to choose the $\bar{F}$ and R_F used for the estimation. No bias force was expected, giving $\bar{F} = 0$. The variance R_F was chosen to be large for the forces and small for the torques.

The high-pass characteristic of the force estimates put limitations on how the assembly could be performed, since it made it impossible to control constant forces. If a force sensor would be used, search motions could be followed by force control to maintain the contact, but when force estimation was used, the contact had to be maintained using position control after the contact force transient was detected. Since the contact torque estimates were unreliable, the rotational search for a contact torque in

state 7 (see page 152) was replaced by a position-controlled motion to the expected final orientation of the shield can. To be able to do this maneuver robustly it had to be assumed that the mounting plane of the PCB was known with good accuracy, which was a reasonable assumption to make, as the PCB was placed in a fixture parallel to the table. Gripping uncertainties, corresponding to small rotations around the $f2$ z-axis, were not expected to be a problem, as the gripper was compliant in this direction and because the shield can was rotated down when in contact with a corner of the socket, such that the shield can was forced onto the socket by its edges.

Force data from an experimental execution is given in Fig. 10.3. The high-pass filtered measurements from the force sensor serve as ground truth for the force estimates. Two versions of the estimated force are shown, with and without a priori information about the low contact torques. The first state shown is the search for contact in the $f1$ z-direction. The transition condition, a large positive z-force can be seen in all force curves at $t = 0.5$ s. The following state was the search in the positive y-direction and it made contact at $t = 0.7$ s, which can be seen by a large negative y-force. State 5 was a search in the x-direction. The search motion was made with contact in both the z- and the y-direction, and this initially caused a friction peak in the x-force (at $t = 0.8$ s) before the actual contact was made at $t = 1.2$ s. The transition to the next state was finally made at $t = 1.2$ s. The final search for the corner of the socket was then made in two steps; first a y-search in state 6 and then an x-search in a new state, here called 6.5. The transition condition for the y-search can be seen at $t = 1.6$ s and the transition condition for the x-search at $t = 1.9$ s. State 7 was the position control of the orientation, such that the shield can was rotated down onto the socket. The rotation was made around the origin of frame $f2$. Modeling errors in the position of this frame was the reason for the large z-forces around $t = 2.7$ s, as the rotation was not made exactly around the origin of $f2$. These forces were detected and the reference position in the z-direction was adjusted. The shield can was pressed onto the socket with a large force in state 8, which can be seen in the z-force at $t = 3.0$ s. Finally, the robot waited 0.3 seconds in state 8.5 and then moved away in state 9.

Measured and estimated contact torques from the experimental execution are given in Fig. 10.4. It can be seen from the sensor measurements that torques significantly different from zero were present only during the last stage of the assembly, i.e., during state 7 and 8. The estimates that were made without a priori information about small torques were very poor, and neither the magnitude nor the shape showed much resemblance with measured data. Also for the estimates that were made *with* a priori information, the shape correlated very poorly with the measured

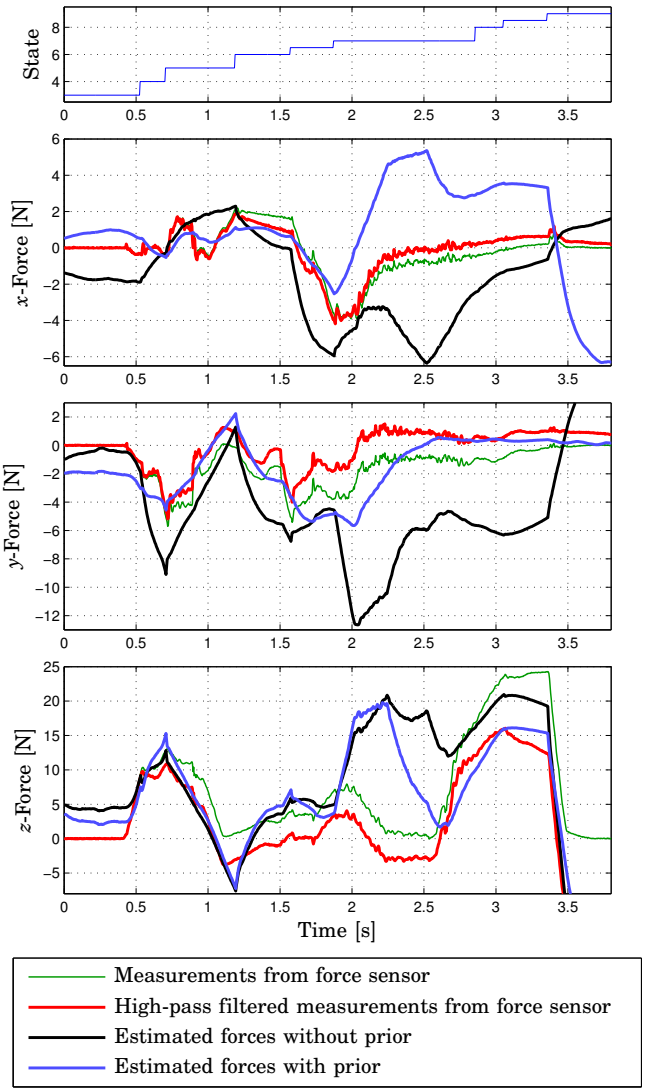


Figure 10.3 Measured and estimated forces from the assembly experiment together with the state sequence. The data from the force sensor is included for comparison and was not used for control. The high-pass filtered measurements from the force sensor were used as ground truth.

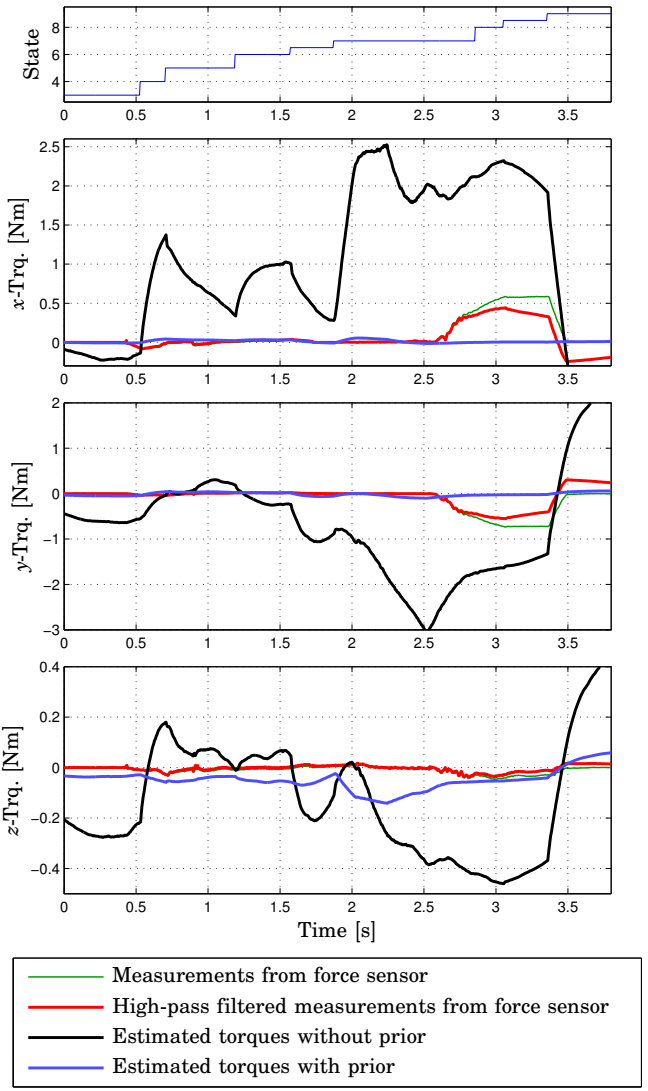


Figure 10.4 Measured and estimated torques from the assembly experiment together with the state sequence. The data from the force sensor is included for comparison and was not used for control. The high-pass filtered measurements from the force sensor were used as ground truth.

torques, and could not be used to detect contact torques. The estimates using the a priori information were, however, much closer to the actual torques, which provided for making the simultaneously estimated contact forces being more accurate than without a priori information.

Discussion

Estimating forces from the joint errors instead of using a force sensor introduces some difficulties in the implementation of the assembly operation. It requires the choice of an appropriate detuning of the joint controllers, and the assembly strategy may have to be altered to not include any control of constant forces. Since the disturbances in the estimates may be quite large, special care must be taken when choosing force thresholds in the design of the assembly sequence.

When the robot was not moving, the Coulomb friction in the joints made it particularly hard to estimate the forces, since the contribution from gravity and other disturbance forces was unknown, and it was very difficult to predict how much additional torque was needed in the various different directions to overcome the friction and to make the joint move. When the robot was moving, however, the Coulomb friction torque was close to constant and even a small external force (e.g., caused by a collision) could affect the motion and be seen as a transient in the joint errors. Since the disturbances from the friction were very similar between different executions of the same motion, the situation became even better and it was possible to robustly detect forces with the same order of magnitude as the friction disturbances. Since the disturbances were velocity dependent, there might, however, be a need to retune the force thresholds if the speed of motion was changed.

The fact that the disturbances to a large extent were systematic, indicates that adaptation or learning techniques could be successful in improving the performance. By further on considering the entire signal instead of its instantaneous value it is probably possible to find more robust transition conditions.

The disturbances on the force estimates increased with the speed. One reason for this could be that the actual joint positions (that were used to estimate the forces) lagged behind the position references at high speeds. Other reasons could be unmodeled dynamic forces or viscous friction.

The detuning of the joint controllers reduced the position accuracy of the robot, but it can also be seen as a way to achieve low-level impedance control, which may be beneficial for contact operations.

For the proposed force-estimation method, no gravity compensation was needed, which is closely coupled to the property that only high-pass filtered torques could be estimated. Not having to compensate for gravity

can be useful in particular when manipulating objects of unknown mass.

In the experiment displayed in Fig. 10.3, the estimated forces were the best during the first 1.9 seconds, when the joint velocities were relatively low and the main disturbance was the joint friction. After $t = 1.9$ s, the estimates were poor in the z-direction up to $t = 2.6$ s and in the x-direction during the rest of the execution. One reason for the large disturbances could be that the assumption of small contact torques around the corner of the shield can was violated from $t = 2.6$ s to $t = 3.5$ s, when the shield can was pressed onto the socket. Another source of the large error could be that some robot joints were moving faster during states 7 and 9, which could cause disturbances as discussed in the previous paragraph. These errors did, however, not affect the performance of the assembly task.

In the shield-can assembly scenario the sensing problem was very hard, since the contact forces were in the same order of magnitude as the disturbances caused by friction in the joints, and using a priori knowledge about the small contact torques around the corner of the shield can turned out to be crucial for the example assembly task. Without it, the transitions triggered by positive x-force at $t = 1.2$ s, negative y-force at $t = 1.6$ s, and negative x-force at $t = 1.9$ s could probably not have been detected properly. In a different scenario, with contact torques larger than the friction disturbances, it should be possible to estimate the torques and perform assembly without making assumptions about small contact torques.

The prior distribution of the contact torques was much smaller than the uncertainties of the contact torque estimates originating from the control errors if no prior was used. Hence, the contact torques estimated using both the control errors and the prior were not much better than what would be achieved by using the prior only. Consequently, using the prior did not make it possible to detect contact torques, but it improved the accuracy of the estimated forces.

The expression (10.3) provides the maximum-likelihood estimate of the forces if all distributions are Gaussian, which is not the case in practice, but in lack of more detailed knowledge about the distributions, Eq. (10.3) serves as a useful approximation.

10.3 Robotic Force Estimation Using Motor Torques and Modeling of Low-Velocity Friction Disturbances

Background and Introduction

Measured motor torques are often available in robots and can be used for force estimation, but they contain large disturbances. An example of measured motor torques from a dual-arm assembly execution is displayed in Fig. 10.5, together with external joint torques derived from a force sensor.

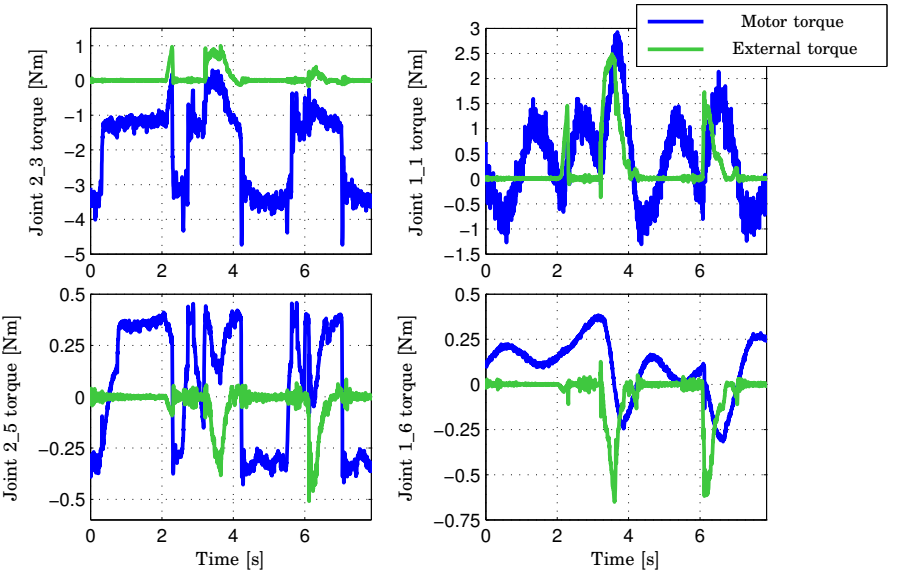


Figure 10.5 Measured motor torque and applied external torque from an assembly sequence. The diagrams in the left column show data from the left arm, which was controlled to move, and the diagrams in the right column show data from the right arm, which was controlled to be still. The upper diagrams show data from base joints, and the lower diagrams show data from wrist joints. The joints chosen for display are those where the external torques were the most visible, thus making it comparatively easy to detect the external torques in the motor torque data. Still, the disturbances were as large or even larger than the signal of interest.

The right robot arm was controlled to be still while the left robot arm was manipulating an object held by the right robot arm. For the left arm you can see the motor torques jumping between two levels corresponding to the magnitude of the Coulomb friction. On top of this binary pattern you can see irregularities resembling the external torques, but they were typically smaller than the friction disturbances. For the right arm, which was controlled not to move, the friction torques were less predictable and the external torques were even harder do distinguish in the signature of the motor torques than for the left arm.

This section presents a method for estimating external forces based on motor torques in the presence of friction disturbances. The focus of the method is not on estimating the friction torques of the individual joints rigorously, but on modeling the velocity-dependent uncertainties in the friction torques and combining measurements from multiple joints to

compute an accurate estimate of the contact force. In particular, it takes the noise in the velocity measurement into account, and models that the Coulomb friction is quite well known when a joint is moving, but has much larger uncertainty for velocities close to zero. The force is estimated by solving a convex optimization problem in real time, and an approximate confidence interval is also calculated.

The validity of the approach was investigated by comparing the estimated forces to measurements from a force sensor. The method was finally tested in a dual-arm screwing assembly task, part of the emergency-stop-button assembly described in Sec. 9.4.

Method

Modeling The method used for force estimation is based on the measured joint motor torques. The model used is

$$\tau_m = \tau_{grav} + \tau_{dynamic} + \tau_{ext} + \tau_e \tag{10.4}$$

where τ_m denotes the measured motor torques, τ_{grav} denotes the torques originating from gravity, $\tau_{dynamic}$ denotes dynamic torques originating from accelerations of the robot, τ_{ext} denotes external torques, and τ_e denotes disturbances due to friction, measurement noise, modeling errors, etc.

The influence from gravity, τ_{grav}, can be calculated if the mass and center of mass are known for each link of the robot. If they are not known, it is fairly easy to perform identification experiments to find these parameters. The actual calculation is described in [Spong et al., 2006, p. 271]. The dynamic torques, $\tau_{dynamic}$, can also be calculated if the dynamic parameters of the robot are known, i.e., moment of inertia for each link of the robot. The dynamic torques will, however, be small in tasks where it is interesting to use force estimation, as the robot will be interacting with the environment and thus needs to move quite slow. It is therefore assumed that the dynamic torques can be neglected.

The external joint torques originate from external forces and torques applied to the robot. If it is assumed that all external forces are applied to the end effector of the robot, the external joint torques are given by

$$\tau_{ext} = J(q)^T F \tag{10.5}$$

where $J(q)$ is the Jacobian of the robot, q the joint coordinates, and F denotes the force/torque applied to the end effector.

Disturbance torques The disturbance, τ_e, influencing each joint mostly consists of Coulomb friction, which can be modeled to give the following

contribution for joint i

$$\tau^i_{Coulomb} = \begin{cases} \tau^i_{C,max}, & \dot{q}_i > 0 \\ \tau^i_{C,min}, & \dot{q}_i < 0 \end{cases} \tag{10.6}$$

where $\dot{q}_i$ denotes the velocity of joint i, and $\tau^i_{C,max}$ and $\tau^i_{C,min}$ denote the constant friction levels for positive and negative velocities respectively. What happens at zero velocity is not given by the model, and the torque may be anywhere in the interval $[\tau^i_{C,min}, \tau^i_{C,max}]$. Therefore, for low velocities close to zero, the Coulomb friction contribution can be modeled by a uniform random variable.

Another type of friction is viscous friction. It can be modeled to give the following contribution for joint i

$$\tau^i_{viscous} = c_i \dot{q}_i \tag{10.7}$$

where c_i is a constant specific for each joint.

Another large disturbance is measurement noise, which can be modeled to have a zero-mean Gaussian distribution.

Disturbance model To find out the disturbance characteristics, an identification experiment was performed for each joint. The joint was then moving back and forth with a low piecewise constant acceleration, without any external forces applied to the robot. Two versions of this experiment are displayed in Fig. 10.6; the upper diagram shows an experiment with low velocities, and the lower diagram an experiment with higher velocities. The raw data, sampled at 250 Hz, were low-pass filtered with the discrete-time filter (10.8) to reduce the influence of the noise.

$$H(z) = \frac{0.4}{1 - 0.6z^{-1}} \tag{10.8}$$

The Coulomb friction is easy to see in both experiments. As was suggested earlier, the amount of friction for zero velocities varies between $\tau_{C,min}$ and $\tau_{C,max}$, and due to noise in the velocity measurements this is true also for measured velocities slightly different from zero. Aside from Coulomb friction, the experiment with large velocities shows viscous friction. Further, there is also noise present.

A probabilistic model of the disturbances is therefore that the Coulomb and the viscous friction are the outcome of a uniform random variable with a velocity-dependent range. This range is zero for large velocities and the range grows for low velocities. One way to describe this range is by using sigmoid functions for describing the upper and the lower limits. To incorporate also viscous friction, a term linear in velocity is added.

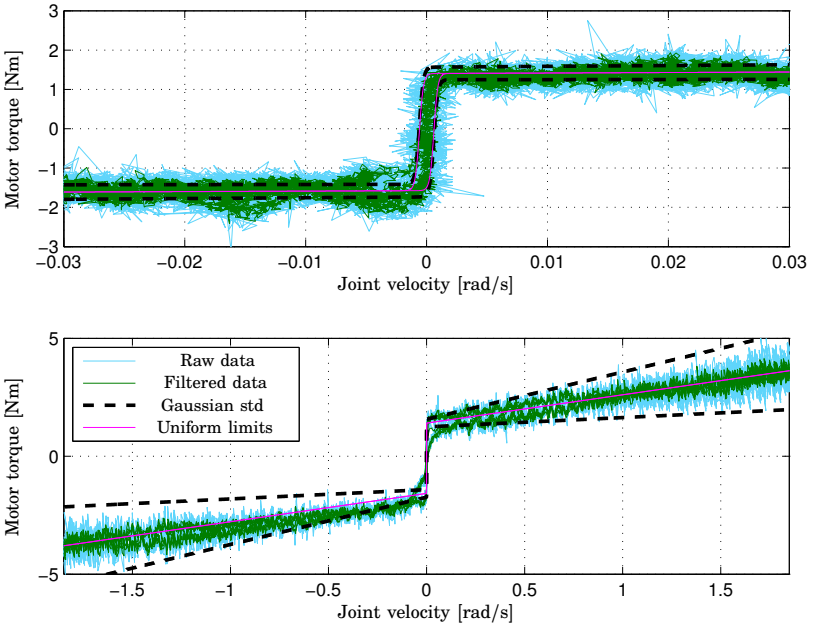


Figure 10.6 Experimental data from an experiment where one joint of the robot was controlled to move back and forth with piecewise constant acceleration. The upper diagram shows an experiment with only low velocities, and the lower diagram shows an experiment with higher velocities. The disturbance characteristics are clearly visible in this experiment. Also shown is one standard deviation of the measurement noise multiplied with a velocity dependent factor, and the upper and lower limits for the uniform distribution describing the Coulomb and the viscous friction.

The upper and lower limit for each joint can be described by the following functions (joint index omitted)

$$\begin{aligned} \tau_{f,max}(\dot{q}) &= \tau_{C,min} + \frac{\tau_{C,max} - \tau_{C,min}}{1 + e^{-A(\dot{q}+B)}} + c\dot{q} \\ \tau_{f,min}(\dot{q}) &= \tau_{C,min} + \frac{\tau_{C,max} - \tau_{C,min}}{1 + e^{-A(\dot{q}-B)}} + c\dot{q} \end{aligned} \tag{10.9}$$

The parameter A determines the slope of the sigmoid function, and the parameter B the width of the area between the curves. Parameters for such functions were manually tuned for each joint of the robot, and an example is seen as magenta curves in Fig. 10.6.

A Gaussian noise term is used to account for measurement noise, uncertainty in the friction limits and unmodeled disturbances. In Fig. 10.6, it can be seen that the variance of the noise increases when the veloc-

ity increases. The model used is therefore that the variance of the noise is velocity dependent and the standard deviation for different velocities is calculated as the standard deviation for low velocities multiplied with a factor $(1 + k|\dot{q}_i|)$. One standard deviation of the noise is displayed in Fig. 10.6. Data recorded during assembly operations indicated that the actual disturbances at high velocities were higher than the measured data in Fig. 10.6 indicate. Hence, the one-standard-deviation limit may appear overly pessimistic in this figure.

To conclude, the total disturbance torque is modeled as

$$\tau_e = \tau_f + e \tag{10.10}$$

where $\tau_{f,min}(\dot{q}) \leq \tau_f \leq \tau_{f,max}(\dot{q})$, and e is zero-mean Gaussian with diagonal covariance matrix $\mathrm{E}[ee^T] = R_e(\dot{q}) = \mathrm{diag}(\mathbf{1} + k|\dot{q}|)^2 R_e(0)$.

Force estimation Let $\bar{\tau}$ be the motor torques compensated for gravity, calculated as

$$\bar{\tau} = \tau_m - \tau_{grav} \tag{10.11}$$

Using (10.4), (10.5), (10.10), and the assumption that the dynamic torques are negligible, this gives

$$\begin{aligned} \bar{\tau} &= \tau_{ext} + \tau_e \\ &= J^T F + \tau_f + e \end{aligned} \tag{10.12}$$

where $\bar{\tau}$ and J are given by measurements, and τ_f and e are random variables with uniform and Gaussian distributions respectively. The ML (Maximum Likelihood) estimate of F is then given by the solution to

$$\begin{aligned} \underset{\text{over } F,\tau_f}{\text{minimize}} \quad & \left(\bar{\tau} - J^T F - \tau_f\right)^T R_e^{-1} \left(\bar{\tau} - J^T F - \tau_f\right) \\ \text{subject to} \quad & \tau_{f,min} \leq \tau_f \leq \tau_{f,max} \end{aligned} \tag{10.13}$$

The estimate given by (10.13) can be improved by adopting a Bayesian approach and using prior knowledge of F in the particular task, just like in Sec. 10.2. The type of prior knowledge that can be used is, for instance, that the contact torques are small compared to the torque disturbances, and by reflecting this knowledge in the distribution of F it is possible to improve the quality of the estimated contact forces.

Assuming that the prior on F is Gaussian with $\mathrm{E}[F] = \bar{F}$ and $\mathrm{E}[(F - \bar{F})(F - \bar{F})^T] = R_F$, and that F and e are uncorrelated, the ML estimate of F is given by the solution to

$$\begin{aligned} \underset{\text{over } F,\tau_f}{\text{minimize}} \quad & \left(\bar{\tau} - J^T F - \tau_f\right)^T R_e^{-1} \left(\bar{\tau} - J^T F - \tau_f\right) \\ & + \left(F - \bar{F}\right)^T R_F^{-1} \left(F - \bar{F}\right) \\ \text{subject to} \quad & \tau_{f,min} \leq \tau_f \leq \tau_{f,max} \end{aligned} \tag{10.14}$$

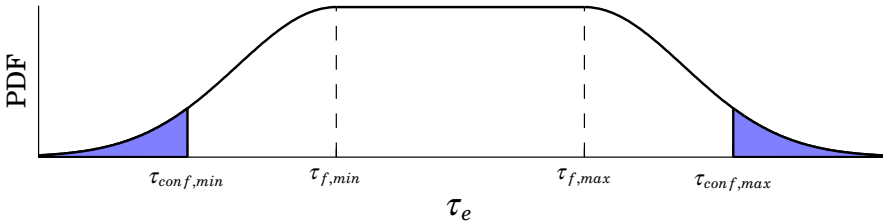


Figure 10.7 Illustration of the proposed confidence interval on the probability density function (PDF) of $\tau_e = \tau_f + e$, a flat part and two Gaussian tails. The blue areas indicate the portion of the measurements expected to be outside the confidence interval.

The problem (10.14) is convex and can be solved numerically in real time, as described on page 189.

Confidence interval estimation The uncertainty of the estimate given by (10.14) varies significantly with the velocity of the different joints and the robot Jacobian, etc. Hence, it is important to calculate the uncertainty of every estimate individually.

It is difficult to compute exact quantiles for the solution of (10.14), but this section describes a method for extracting approximate confidence intervals that can be computed in real time. The method is first described for the case with a single robot joint without prior, and then generalized to handle multiple joints and a prior distribution on F.

Confidence interval estimation — One-dimensional case The proposed confidence interval for the case of a single robot joint with no prior and the Jacobian $J = 1$ is illustrated in Fig. 10.7. The limits are calculated as

$$\begin{aligned} \tau_{conf,min} &= \tau_{f,min} - \lambda\sigma \\ \tau_{conf,max} &= \tau_{f,max} + \lambda\sigma \end{aligned} \tag{10.15}$$

where σ is the standard deviation of the Gaussian tails and λ is a parameter deciding the confidence level of the confidence interval.

For the special case where $\tau_{f,min} = \tau_{f,max}$ (the distribution of τ_e is Gaussian), the portion of the measurements outside the confidence interval is $2(1 - \Phi(\lambda))$, where $\Phi(\cdot)$ is the cumulative distribution function of the zero-mean unit-variance Gaussian distribution.

An alternative way of finding the limits (10.15), is to minimize the negative log-likelihood function of τ_e and adding a gradient to push the solution toward the upper or lower limit. The log-likelihood of a zero-mean

Gaussian with standard deviation σ is given by

$$\log \mathcal{L}(e) = -\frac{e^2}{2\sigma^2} + \text{const.} \tag{10.16}$$

$$\frac{d}{de}\left(\log \mathcal{L}(e)\right) = -\frac{e}{\sigma^2} \tag{10.17}$$

Hence, at the limits of the confidence interval, the derivative of the negative log-likelihood function of τ_e is

$$-\frac{d}{de}\left(\log \mathcal{L}(\pm\lambda\sigma)\right) = \pm\frac{\lambda}{\sigma} \tag{10.18}$$

Consequently, adding a gradient with one of the slopes (10.18) to the negative log-likelihood of τ_e and finding the minimum, gives one of the limits (10.15) as the solution. This way of calculating the limits is described because it generalizes to higher dimensions better than (10.15).

Confidence interval estimation — Multi-dimensional case For the multi-joint problem (10.14), first assume that $\tau_{f,min} = \tau_{f,max}$, (i.e., Gaussian distribution). The standard deviation σ of the marginal distribution of F in the direction of the unit vector v is then given by

$$\sigma = \sqrt{v^T \left(JSR_e^{-1}J^T + R_f^{-1}\right)^{-1} v} \tag{10.19}$$

where S is the identity matrix for the Gaussian case but may have other values for the general case, as described later in this section. The limits of the confidence interval in the direction v are then given by the F solving

$$\begin{aligned} \underset{\text{over } F,\tau_f}{\text{minimize}} \quad & \left(\bar{\tau} - J^T F - \tau_f\right)^T R_e^{-1} \left(\bar{\tau} - J^T F - \tau_f\right) \\ & + \left(F - \bar{F}\right)^T R_F^{-1} \left(F - \bar{F}\right) \mp \tfrac{\lambda}{\sigma} v^T F \\ \text{subject to} \quad & \tau_{f,min} \le \tau_f \le \tau_{f,max} \end{aligned} \tag{10.20}$$

where the "$-$" in the "$\mp$" is for the upper limit, and the "$+$" is for the lower limit. This formulation is obtained by adding the gradient (10.18) to the problem (10.14).

Returning to the general case, when $\tau_{f,min} \neq \tau_{f,max}$, some of the joints may get an estimated τ_e in the range $\tau_{f,min} < \tau_e < \tau_{f,max}$. The cost function for that joint is then locally flat, cf. Fig. 10.7, and should not be considered when calculating (10.19).

To find out for which joints the estimated τ_e ends up in the Gaussian part of the distribution, we propose the following algorithm. It is assumed that n joints are used for force estimation and that R_e is diagonal.

1. Set $S = 0_{n \times n}$.
2. Calculate (10.19).
3. Solve (10.20).
4. For the joints where $\tau_f = \tau_{f,min}$ or $\tau_f = \tau_{f,max}$, set the corresponding diagonal elements of S to 1.
5. If S was modified in step 4), go to step 2). Else quit.

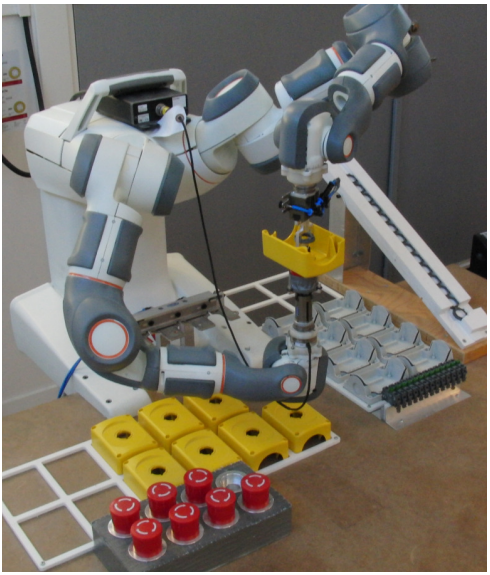
The intuition behind the above algorithm is the following. The problem (10.20) is first solved using a gradient based only on the prior. If the Coulomb friction for all joints is large, the resulting τ_e may all be within the flat part of the distribution (cf. Fig. 10.7) and only the prior is used for determining the confidence interval. If any of the estimated τ_e reaches the Gaussian parts of the distributions, the gradient based only on the prior will not be able to push the estimate sufficiently far down the Gaussian tails. The value of σ in (10.19) is then modified to include all joints where the τ_e estimate is in the Gaussian part, resulting in a steeper gradient, which may in turn push the estimate of additional joints to the Gaussian part of the distribution. The process (steps 2–5) is iterated until convergence.

Implementation The optimization problem (10.14) is a convex optimization problem of fairly small size and can be solved in real time in a reliable manner. To this purpose, CVXGEN [Mattingley and Boyd, 2012] was used. It is a code generator for embedded convex optimization. The generated code is library-free C code, and this code was connected to the robot controller via an Ethernet connection.

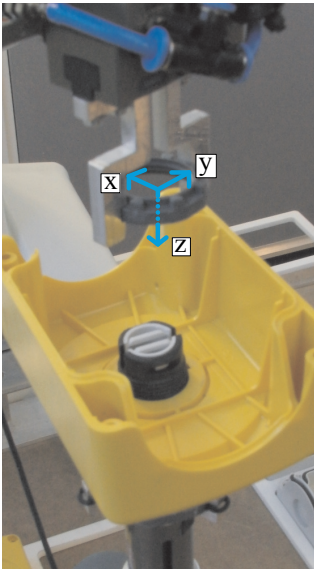
The generated solver was run on a Linux PC and the computation time to arrive to a solution was in the order of 0.3 ms. The robot controller was run with a sampling time of 4 ms, and the speed of the solver was therefore sufficient to be run in each sample. The solution with the Ethernet connection introduced a delay of one sample, as the indata to the solver was sent one sample before the solution was returned.

Assembly Scenario

A part of the assembly of the emergency stop button described in Sec. 9.4 was considered to illustrate the use of the force-estimation method. The subtask considered was the attachment of the button onto the box by screwing a nut. This assembly operation was performed in a dual-arm setting, with one arm holding the button with the yellow box on, and the other arm performing the screwing (2.5–3.5 revolutions). Figure 10.8



(a) Overview of the workcell. The force/torque sensor, mounted at the wrist of the robot's right arm, was only used for verification and evaluation of the estimated forces.



(b) Zoom in and illustration of the frame in which the contact forces were measured.

Figure 10.8 The setup for the assembly task used in the experiments.

shows the robot cell just before the assembly operation was started. The task was modeled and specified using the iTaSC framework [De Schutter et al., 2007], described in Sec. 9.2.

The arm holding the button was *static*, which means that it was controlled not to move, but forces applied to it could cause it to move slightly, since a small joint-position error was needed before the controller could act upon it.

The assembly sequence consisted of first putting the nut on the thread of the button, using the contact forces to find the hole of the nut, though the exact grip of the button was uncertain. Before the nut was properly screwed onto the thread, large side forces could be caused by a bad grip of the nut. If such a force was detected during the first two revolutions, the gripper was opened and re-closed, which usually gave a more centric grip. Due to the way that the nut was gripped, large side forces were generated when the nut was tightened, which could be used to detect when the screwing operation was completed. More details can be found in Sec. 9.4.

The robot used in the assembly scenario was the ABB FRIDA [Kock et al., 2011]. It is a dual-arm manipulator with 7 joints in each arm, devel-

oped for automation of assembly operations. The robot was controlled with the IRC5 control system, extended with an open control system [Blomdell et al., 2005; Blomdell et al., 2010], which made it possible to modify the references for the low-level joint control loops. The motor torques used for force estimation were calculated from the motor currents. The ATI Mini40 force sensor, which was normally mounted under a fixture on the table, was for this experiment mounted on the wrist on one of the arms to give ground truth force data, see Fig. 10.8(a).

Experimental Results

Calibration An experiment where the robot was programmed to slowly move around in its work space was performed to identify the parameters used for calculating the gravity torque, τ_{grav}. The resulting parameters resulted in a mean absolute error ranging from 0.05 Nm for the wrist joints to 0.3 Nm for the base joints.

The friction parameters were tuned by performing experiments of the type that was described on page 184.

Force estimation The force estimation method was tested in an experiment where forces were applied to a static arm (controlled not to move). The estimated forces and confidence intervals are displayed in Fig. 10.9, together with ground truth data from the force sensor. All confidence intervals displayed in this chapter were estimated with $\lambda = 1.96$ in (10.20), which would give a 95 % confidence interval for a Gaussian random variable. The parameter k was set to 5 s/rad.

Fig. 10.9 shows that the estimated force tracked the measured force well, but the confidence intervals seem to be overly pessimistic. The Coulomb friction was, however, a very large disturbance for low velocities. When the robot was moving, the uncertainty in the Coulomb friction was much smaller, as can be seen on the magenta-colored curve in the upper diagram of Fig. 10.6. Large external forces made the robot move slightly, and this gave significantly tighter confidence intervals than when the robot was still, as can be seen, for example, on the z-force at $t = 15$ s and $t = 18$ s in Fig. 10.9.

Screwing assembly task Estimated and measured forces from an execution of the screwing assembly task are displayed in Fig. 10.10. The forces are given in the coordinate frame illustrated in Fig. 10.8(b). It can be seen that the estimated forces tracked the measured forces, at least when the measured forces were non-zero, i.e., during contact operations. When the measured forces were zero, however, the estimated forces were sometimes a bit wrong, e.g., in the y- and the z-directions around $t = 1.6$ s. These estimation errors were most likely due to modeling errors, as the robot was moving quite fast in this part of the assembly, but it was known that

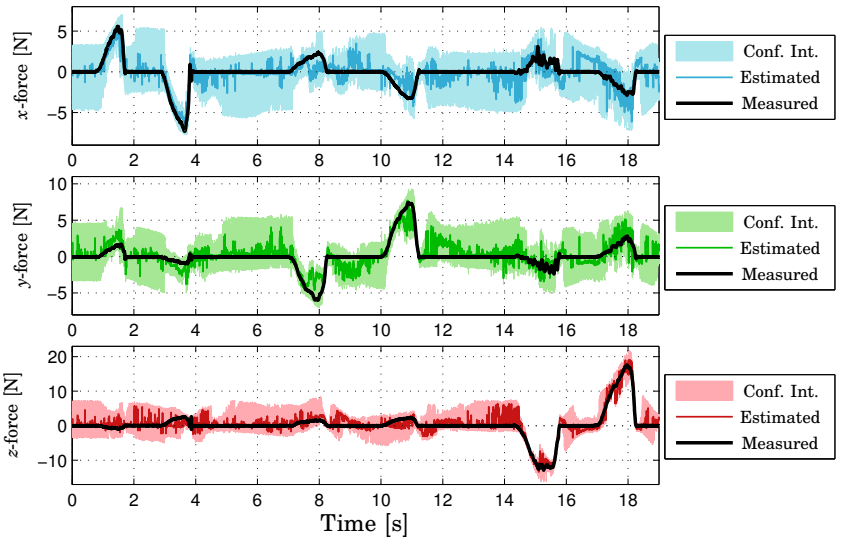


Figure 10.9 Data from an experiment where forces were applied to the end effector of a static arm. A wrist-mounted force sensor was used to measure a ground truth, and data from this are shown with black lines. The estimated force is displayed together with a confidence interval.

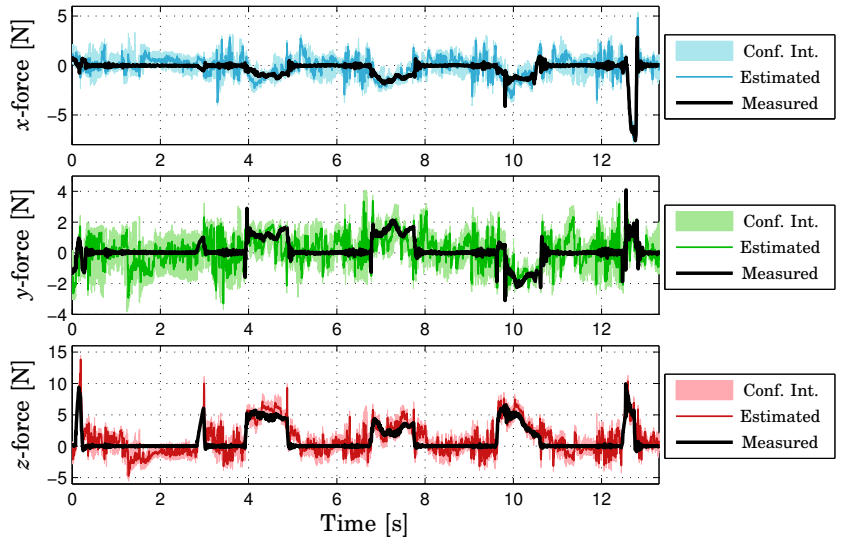


Figure 10.10 Measured and estimated contact forces from an execution of the screwing assembly task. The forces are given in the coordinate frame illustrated in Fig. 10.8(b). The computed confidence intervals are also shown.

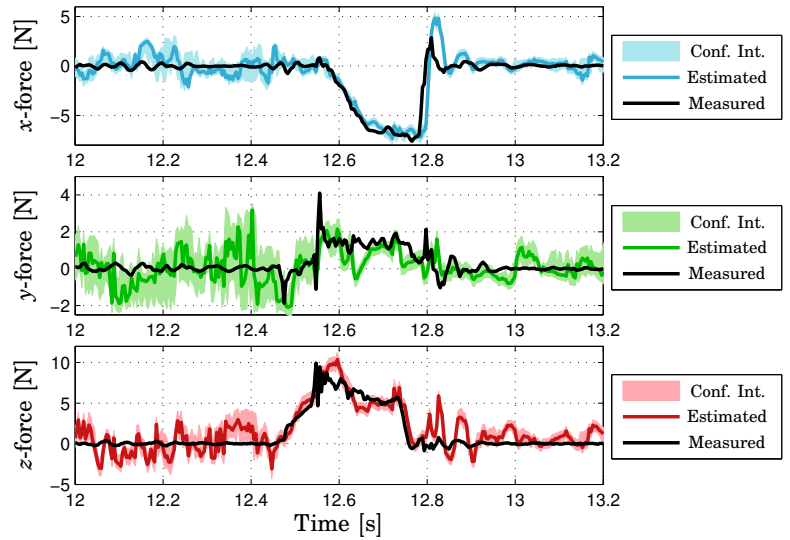


Figure 10.11 A zoom in on the measured and estimated force data from Fig. 10.10.

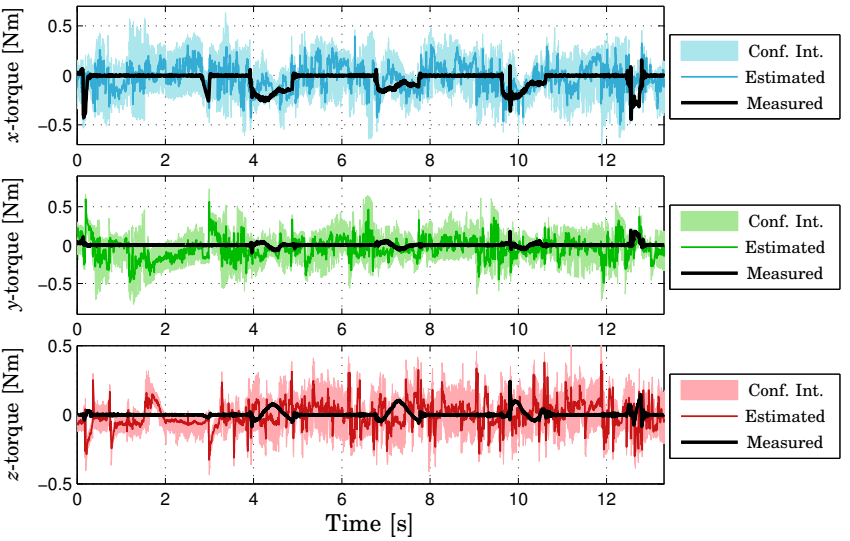


Figure 10.12 Measured and estimated contact torques from an execution of the screwing assembly task. The torques are given in the coordinate frame illustrated in Fig. 10.8(b). The computed confidence intervals are also shown.

the motion would be performed in free space when the robot was moving fast, and therefore it was not crucially important to get a perfect force estimate.

For the assembly task, some of the important forces to detect were the contact forces in the z-direction when the nut was put on the thread. They were correctly detected at $t = 0.2$ s and $t = 3$ s, and the confidence interval was tight at these moments. The screwing was finished when a large side force was detected at $t = 12.6$ s; a zoom in on this part of the data is displayed in Fig. 10.11. It can be seen that the force estimate was both quite correct and confident when the forces occurred. Some oscillations can be seen in the force estimate, e.g., in the z-force around $t = 12.2$ s. This might be caused by unmodeled disturbances, like cogging torques in the motors or mechanical resonances.

The estimated contact torques together with those measured with the force sensor are displayed in Fig. 10.12. For most samples, the confidence intervals of the estimated contact torques included the zero torque, since the friction torques were very large in comparison to the contact torques. Consequently, larger contact torques would be required for the estimator to be able to detect them reliably.

The force estimates presented in Figs. 10.10-10.12 were calculated using data from both arms, and in Table 10.1 they are compared to estimates based on only the right (static) or left (moving) arm. It can be seen that the estimate using the static arm, gave the most measurements within the confidence interval. This data should, however, only be used to evaluate the quality of the confidence intervals, not the force estimates. When the joints were not moving, the large uncertainty in the Coulomb friction resulted in wide confidence intervals, and hence many measurements were inside the confidence intervals. When the robot was moving, however, the model seemed to be a bit too confident about the estimate. Only looking at the samples when external forces were present did not change much.

The lower part of Table 10.1 shows the mean absolute estimation error. Here it can be seen, that when all samples were considered, the estimates from the static arm were the best, but only slightly better than the estimate using both arms. It may be surprising that using only one arm can give better results than using both arms, but when only the static arm was used, the confidence interval was large and usually enclosed the prior, which was $\bar{F} = 0_{6\times1}$ in our example. Hence most estimates were pulled to the prior, which was almost equal to the actual forces for most samples in this particular sequence. When there were external forces present, which is the situation when the force estimation is really useful, the estimates based on both arms were significantly better than those based on any single arm, as seen in the lower right part of Table 10.1.

	All samples			Samples when external force was non-zero		
	Percentage of ground truth force/torque measurements within confidence interval					
Wrench component	Static arm	Moving arm	Both arms	Static arm	Moving arm	Both arms
x-force	96.2 %	66.7 %	78.6 %	91.2 %	65.2 %	70.7 %
y-force	95.6 %	64.4 %	82.4 %	90.0 %	64.2 %	74.8 %
z-force	91.7 %	61.9 %	58.4 %	81.2 %	67.5 %	57.2 %
x-torque	97.1 %	87.4 %	86.9 %	93.2 %	64.7 %	80.5 %
y-torque	97.9 %	80.5 %	72.2 %	95.3 %	85.8 %	78.4 %
z-torque	99.8 %	72.8 %	67.7 %	99.6 %	89.9 %	77.5 %
	Mean absolute error					
Wrench component	Static arm	Moving arm	Both arms	Static arm	Moving arm	Both arms
x-force	0.60	1.08	0.60	1.05	1.23	0.62
y-force	0.63	1.40	0.64	1.14	1.42	0.64
z-force	0.91	1.38	1.09	1.33	1.29	1.02
x-torque	0.066	0.22	0.095	0.16	0.25	0.096
y-torque	0.036	0.15	0.11	0.073	0.13	0.099
z-torque	0.014	0.050	0.074	0.036	0.057	0.091

Table 10.1 Statistics for the force estimation in the screwing assembly task. The upper half shows the percentage of the samples where the measured force was within the computed confidence interval. The lower half shows the mean absolute estimation error (forces in N, torques in Nm). The left part shows statistics for when the entire assembly sequence is considered, while the right part only considers those samples when the measured force was non-zero, i.e., when the arms were in contact.

Discussion

Experiments showed that, when the arms were in contact, complementing motor torque data from a moving arm with data from a static arm significantly improved the quality of the estimated forces. The static arm could not have been exploited properly with previously published force estimation methods, which do not account for the uncertainty of the Coulomb friction at low velocities.

Using a prior distribution on F with small variance on the contact torques, can be seen as putting a soft constraint on the contact torques. Instead of estimating a 3-DOF force and a 3-DOF torque, the problem is then almost reduced to estimating only a 3-DOF force. The increased redundancy in the problem gives a better force estimate.

The prior distribution on F was chosen to be Gaussian, which is a really crude approximation of the true distribution. Using a Gaussian prior, however, leads to fast calculations and can be a useful approximation when there is not much information about the true distribution available. The variance of the prior should be chosen according to process knowledge; how large forces and torques that are expected.

The magnitudes of the contact torques used in this chapter were small compared to the uncertainties in the torque estimates. This was mainly caused by the relatively large disturbances. The contact torques considered were in the order of 0.1–0.3 Nm, which was in the same order of magnitude as the errors in the gravity torque compensation. Also errors in the Coulomb friction modeling gave disturbances in the same order of magnitude as the contact torques. Estimating forces was more advantageous, as relatively small forces could give rise to relatively large joint torques through long lever arms.

There is no fundamental limitation of the method preventing estimation of contact torques. In an application where the contact torques are larger compared to the torque disturbances the full 6D force/torque could be estimated.

Some of the parameters used for force estimation were manually tuned, including A and B in (10.9), the low-pass filter, and the velocity dependence of the Gaussian noise term. This should be possible to do in an automatic fashion, i.e., make experiments of the type presented on page 10.3 and choose the parameters by optimizing some criterion. This would further simplify the use of the method and it is considered some of the future work.

10.4 Conclusions

This chapter presents two approaches for estimating forces without any force sensor, and both methods were experimentally validated in small-part assembly tasks. The estimated forces had larger disturbances than forces measured by a force sensor, but due to the large cost of force sensors the force estimation methods can still be relevant for practical applications.

The method presented in Sec. 10.2 requires detuning of the joint controllers, which reduces the position accuracy of the robot, but the reduced stiffness can be advantageous for contact operations. The robot model is very simple and does not require any gravity compensation, but on the other hand only high-pass filtered forces are estimated, making it impossible to measure constant forces.

The method based on motor torques, presented in Sec. 10.3 models the robot more carefully and gives more reliable force estimates. It also provides confidence intervals, which is particularly useful since the disturbances vary significantly with the joint speeds. Further, there is no need to detune the joint controllers. The detuning is not necessarily a bad thing, but with the method based on motor torques the detuning is optional and can be done without affecting the force estimates. A drawback compared to the method based on control errors is that gravity compensation is needed.

Conclusions

11

Conclusions

11.1 Vision-Based Control

This thesis presents work on how fast and accurate vision can be used for reactive robot control, demonstrated by the realization of a ball-catching robot. Compared to ball-catching robots developed in other labs, my system demonstrated better spatial accuracy and faster response time.

The color of an object provides useful information to the image analysis, but color-based methods are sensitive to varying illumination conditions. The thesis presents a way of designing a constant classifier that can identify an object based on its color for a wide range of illumination conditions.

Objects that are moving have a risk of causing motion blur in images, in particular in conditions of moderate lighting intensities. A very fast method for detecting where motion blur occurred is proposed. By means of this method it was possible to improve how accurately the positions and sizes of thrown balls could be estimated in images.

A new way of initializing the Kalman filter has been presented, making it possible to calculate a state estimate that without any need for information about the initial value of the state. Instead the estimate can be determined completely based on the first measurements. This was accomplished by choosing a new state representation where the directions with infinite variance were orthogonal to as many basis vectors as possible.

A strategy for tracking dynamical objects with computer vision was described. It provides a simple way to fuse data from an arbitrary number of pictures captured simultaneously. In combination with a dynamical model of the tracked object, it also enables determination of the position of the object in 3D space without any two images being captured at the same time. Furthermore, it also allows tracking of objects in 3D space, using only a single static camera.

A method for generating trajectories with dynamically updated target points was described. It pays attention to the deadline and uses the smallest acceleration possible to reach the goal in time. When a new target point is received, the trajectory is smoothly modified in real time during the robot motion. The effect of jerk limitation on trajectory generation was also investigated.

Whereas the direct practical applications of a ball-catching robot may be limited, it serves to demonstrate the potential and capabilities of robots. If robots should take the step out of the controlled factory to more dynamic environments they need perception and the capability to react quickly to impressions. For such applications, parts of the knowledge gained from developing a ball-catching system could be reused.

11.2 Force Control and Estimation

The thesis presents work on how force sensing can be used for robotic assembly. Two use cases were implemented to evaluate the feasibility of the approach. It was shown that force sensing allowed the robot to perform assembly operations that would not have been possible using only position control.

Two methods for performing robotic force estimation without any force sensor were presented and used to successfully perform assembly operations. The estimated forces were not as good as the measurements one could get from a force sensor, but on the other hand, a force sensor costs a significant amount of money. Considering the tradeoff between cost and performance, force estimation can, hence, be a competitive option.

Though force sensing expands the domain of tasks that robots can perform, there is still a long way until robots can perform all assembly operations with the same speed and robustness as humans. When considering that robots can work around the clock, the production rates were comparable, but the robotic assembly system still lacked in robustness.

One factor limiting the assembly speed of the system used was that it could not perform compliant motions with the same performance as humans. Though this is one important limitation, I believe that the main problem to solve before force-based assembly becomes widely adopted in industry is the task specification. The assembly operations described in Chapter 9 were developed through trial and error, and they required tuning of force thresholds and control parameters. Over time the most common error cases were identified, and the assembly strategies were manually updated to detect and recover from these errors.

Humans can learn diverse assembly tasks with a small amount of task-specific information and can recover from small errors in a natural

way. Until robots come close to these capabilities, humans will still be needed in many assembly operations, but the use of force control expands the set of tasks that can be automated.

11.3 Concluding Remarks

This thesis treats two classes of robot control based on sensor data from the work space: vision and force sensing, which both are important elements for the progress of cognitive robotics.

Computer vision has the benefit that it can give information about a large work volume without coming in contact with or affecting the environment, but has lower spatial accuracy than many other measurement methods. Force sensing can be used to perform mechanical work or accurately infer contact situations, but it is less advantageous in the presence of very large position uncertainties. To quickly find an object in a large volume by searching for a contact force, the robot has to move fast, but high speeds also result in undesirable high collision impact forces when contact with the object is made, due to inertia and control delays.

Here, the two classes of sensor data were treated separately. By continuing the work on combining their respective strengths, the performance of robots can be increased further.

A

Fundamentals of Robotics and Computer Vision

This chapter describes a few concepts from robotics and computer vision that are used in this thesis.

A.1 Homogeneous Coordinates

Rigid Transformations

Assume that you have a point in 3D space described by the coordinate vector $[X \;\; Y \;\; Z]^T$. It can then be described by the homogeneous coordinate vector

$$\mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{A.1}$$

Let $\mathbf{X_a}$ be the coordinate of a point in frame O_a, and let $\mathbf{X_b}$ be the coordinate of the same point in frame O_b. These homogeneous coordinate vectors are then related by a transformation matrix, T, by

$$\begin{bmatrix} X_a \\ Y_a \\ Z_a \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} R_{3\times3} & t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}}_{T} \begin{bmatrix} X_b \\ Y_b \\ Z_b \\ 1 \end{bmatrix} \tag{A.2}$$

where t is the vector from O_a to O_b, described in the coordinate frame of O_a, and R is a rotation matrix defined by

$$R = \begin{bmatrix} e_x & e_y & e_z \end{bmatrix} \tag{A.3}$$

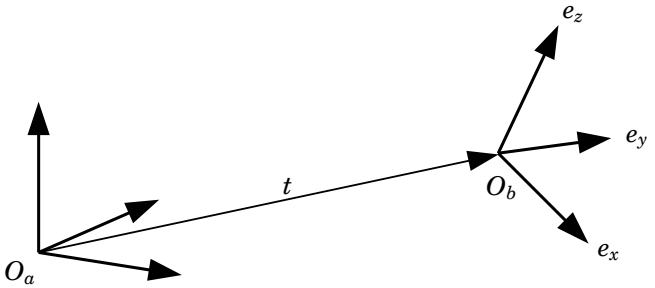


Figure A.1 Illustration of the frames used to describe transformation matrices.

where e_x, e_y, and e_z are the basis vectors of frame O_b described in the frame of O_a, cf. Fig. A.1. The rotation matrix has the properties $R^T R = I$ and $\det(R) = 1$.

By means of homogeneous coordinates and transformation matrices, rigid transformations (rotation + translation) can be performed as linear operations on the form (A.2).

Projective Geometry

This section gives a short introduction to camera modeling and the use of homogeneous coordinates in projective geometry. More details can be found in, for example, [Ma et al., 2003].

When using homogeneous coordinates in projective geometry [Möbius, 1827; Graustein, 1930] it is convenient to introduce the relation "$\sim$", which is defined according to

$$\begin{aligned} &\mathbf{x_1} \sim \mathbf{x_2} \iff \\ &\mathbf{x_1} = \lambda \mathbf{x_2} \text{ for some } \lambda \neq 0 \end{aligned} \tag{A.4}$$

where $\mathbf{x_1}$ and $\mathbf{x_2}$ are homogeneous coordinate vectors. If Eq. A.4 is fulfilled, $\mathbf{x_1}$ and $\mathbf{x_2}$ represent the same point, even if $\lambda \neq 1$. To recover the point that is represented by a homogeneous coordinate vector, scale it so that its last element becomes equal to 1. To illustrate this, consider a 2D example:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \sim \begin{bmatrix} x_1/x_3 \\ x_2/x_3 \\ 1 \end{bmatrix} \tag{A.5}$$

In (A.5), $[x_1 \;\; x_2 \;\; x_3]^T$ is a homogeneous coordinate representation of the 2D coordinate $[x_1/x_3, \;\; x_2/x_3]$.

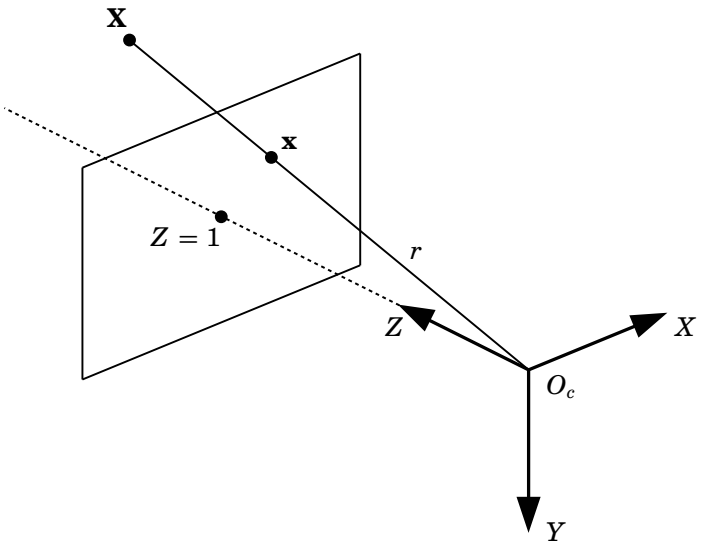


Figure A.2 Illustration of normalized coordinates for pin-hole camera.

Pin-hole camera Let $\mathbf{X}$ be the coordinate of a 3D point, expressed in the coordinate frame of a pin-hole camera, O_c, cf. Fig. A.2. The origin of O_c is the focal point of the camera. Let r be the line that intersects the focal point and the point $\mathbf{X}$. The normalized image coordinate of the projection of $\mathbf{X}$ is defined as the X-Y coordinate where r intersects the plane $Z = 1$. Furthermore, let $\mathbf{x} \sim [x \;\; y \;\; 1]^T$ denote the normalized image coordinate of the projection of $\mathbf{X}$. Then, $\mathbf{x}$ is given by

$$\mathbf{x} = \begin{bmatrix} X/Z \\ Y/Z \\ 1 \end{bmatrix} \sim \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{A.6}$$

This shows that the normalized homogeneous 2D coordinate of an image point is proportional to the coordinate vector of the corresponding 3D point.

Equation (A.6) is valid only for normalized image coordinates, and $\mathbf{X}$ must be given in the camera coordinate frame. A more general expression

is given by

$$\underbrace{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}_{\mathbf{x}} \sim \underbrace{\underbrace{\begin{bmatrix} \gamma f & sf & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} R_{3\times3} & t_{3\times1} \end{bmatrix}}_{G}}_{P} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{\mathbf{X}} \tag{A.7}$$

The matrix G contains the extrinsic coordinates, depending on the pose of the camera, and defines a rigid transformation (cf. Eq. (A.2)), transforming the 3D point to the coordinate system of the camera. The matrix K contains the intrinsic parameters. They consist of an offset, $[x_0, y_0]$, a scaling, f, a skewness parameter, s, and a parameter telling how rectangular the coordinate system is, γ.

To conclude, the properties of a pin-hole camera can be captured in a matrix, $P \in \mathbb{R}^{3\times4}$. A 3D point, $\mathbf{X}$, and its camera projection, $\mathbf{x}$, are related by the equation

$$\mathbf{x} \sim P\mathbf{X} \tag{A.8}$$

Lines and Planes

Homogeneous coordinates can be used to define lines and planes in a convenient way.

Let the coordinates in 2D space be described by a homogeneous coordinate vector:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{A.9}$$

A vector $\mathbf{l} = [l_1 \;\; l_2 \;\; l_3]^T$ can then be used to specify a line as all points satisfying

$$\mathbf{l}^T\mathbf{x} = 0 \iff \tag{A.10}$$

$$\begin{bmatrix} l_1 & l_2 & l_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0 \tag{A.11}$$

The vector $[l_1 \;\; l_2]$ is normal to the line $\mathbf{l}$.

Similarly, a vector $\boldsymbol{\pi} = [\pi_1 \;\; \pi_2 \;\; \pi_3 \;\; \pi_4]^T$ can be used to specify a plane

in 3D space as all points $\mathbf{X}$ satisfying

$$\boldsymbol{\pi}^T\mathbf{X} = 0 \iff \tag{A.12}$$

$$\begin{bmatrix} \pi_1 & \pi_2 & \pi_3 & \pi_4 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = 0 \tag{A.13}$$

The vector $[\pi_1 \;\; \pi_2 \;\; \pi_3]$ is normal to the plane $\boldsymbol{\pi}$.

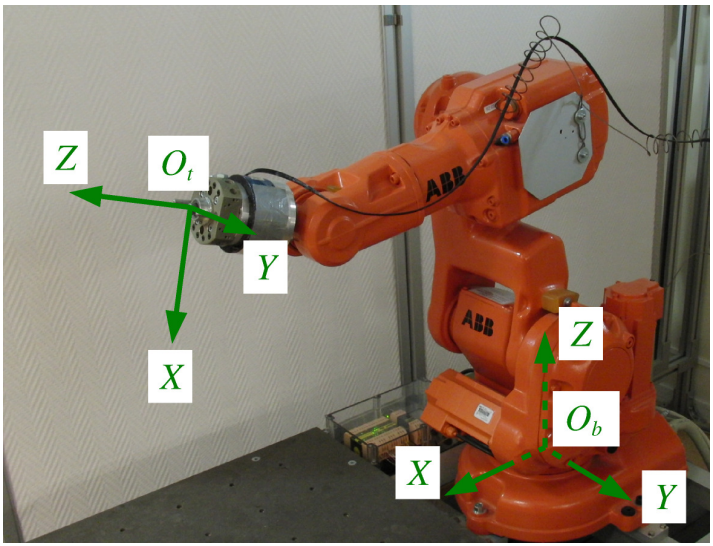


Figure A.3 Illustration of base frame, O_b, and tool frame, O_t, for a robot.

A.2 Robot Kinematics

This section gives a short introduction to robot kinematics. More details can be found in, for example, [Spong et al., 2006].

For a serial robot with revolute joints, let q denote the joint angles of the robot arm:

$$q = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix} \tag{A.14}$$

where n is the number of joints. In Fig. A.3, O_b denotes the robot base frame and O_t denotes the tool frame.

The *forward kinematics* describes the pose of the tool frame in the coordinate system of the base frame as a function of the joint coordinates, q. The relation can be described by a homogeneous transformation matrix, $T(q)$, on the form given in (A.2).

For the *inverse kinematics* problem, the pose of O_t and the corresponding transformation, T_0, are known. The task is to find a q that satisfies $T(q) = T_0$. Depending on the kind of robot, the inverse kinematics problem may or may not have an analytic solution, and there are usually more than one valid solution.

Let $v \in \mathbb{R}^{3\times1}$ and $\omega \in \mathbb{R}^{3\times1}$ be the translational and rotational velocities of O_t, respectively. The robot *Jacobian*, $J(q)$, relates the velocity of the tool frame to the joint velocities according to

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = J(q)\dot{q} \tag{A.15}$$

The Jacobian can also be used to relate the joint torques to the force and torque in the tool frame according to

$$\tau = J(q)^T \begin{bmatrix} F \\ M \end{bmatrix} \tag{A.16}$$

where $\tau \in \mathbb{R}^{n\times1}$ are the joint torques, $F \in \mathbb{R}^{3\times1}$ is the tool force vector, and $M \in \mathbb{R}^{3\times1}$ is the tool torque vector.

Bibliography

ABB (2013). *Robots*. URL: http://www.abb.com/product/us/9AAC100735.aspx.

Albu-Schäffer, A., S. Haddadin, C. Ott, A. Stemmer, T. Wimböck, and G. Hirzinger (2007). "The DLR lightweight robot: design and control concepts for robots in human environments". *Industrial Robot: An International Journal* **34**:5, pp. 376–385.

Alcocer, A., A. Robertsson, A. Valera, and R. Johansson (2003). "Force estimation and control in robot manipulators". In: *Proc. 7th IFAC Symp. Robot control (SYROCO'03)*. Wrocław, Poland, pp. 31–36.

Årzén, K.-E. (2002). "JGrafchart: sequence control and procedure handling in Java". In: *Proc. Reglermöte 2002*. Linköping, Sweden.

Balkenius, C., A. J. Johansson, and A. Balkenius (2003). "Color constancy in visual scene perception". *Lund University Cognitive Science, Lund, Sweden*. ISSN: 1101-8453.

Ballard, D. (1981). "Generalizing the hough transform to detect arbitrary shapes". *Pattern Recognition* **13**:2, pp. 111–122. ISSN: 0031-3203. DOI: http://dx.doi.org/10.1016/0031-3203(81)90009-1. URL: http://www.sciencedirect.com/science/article/pii/0031320381900091.

Barteit, D., H. Frank, and F. Kupzog (2008). "Accurate prediction of interception positions for catching thrown objects in production systems". *Proc. IEEE Intl. Conf. Industrial Informatics (INDIN), Daejon, Korea*, pp. 893–898.

Basler (2005). *Basler A600f User's Manual, Document number DA00056107*. URL: http://www.baslerweb.com/beitraege/unterbeitrag_en_23042.html.

Bätz, G., A. Yaqub, H. Wu, K. Kuhnlenz, D. Wollherr, and M. Buss (2010). "Dynamic manipulation: nonprehensile ball catching". In: *Control Automation (MED), 2010 18th Mediterranean Conference on*, pp. 365–370. DOI: 10.1109/MED.2010.5547695.

Bäuml, B., T. Wimböck, and G. Hirzinger (2010). "Kinematically optimal catching a flying ball with a hand-arm-system". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2010)*, pp. 2592–2599. DOI: 10.1109/IROS.2010.5651175.

Bäuml, B., O. Birbach, T. Wimböck, U. Frese, A. Dietrich, and G. Hirzinger (2011a). "Catching flying balls with a mobile humanoid: system overview and design considerations". In: *Humanoid Robots (Humanoids2011), 11th IEEE-RAS International Conference on*, pp. 513–520. DOI: 10.1109/Humanoids.2011.6100837.

Bäuml, B., F. Schmidt, T. Wimböck, O. Birbach, A. Dietrich, M. Fuchs, W. Friedl, U. Frese, C. Borst, M. Grebenstein, O. Eiberger, and G. Hirzinger (2011b). "Catching flying balls and preparing coffee: mobile humanoid Rollin' Justin perfoms dynamic and sensitive tasks". In: *Proc. IEEE Intl. Conf. Robotics and Automation (ICRA2011), May 9-13, Shanghai, China*, pp. 3443–3444.

Birbach, O. and U. Frese (2009). "A multiple hypothesis approach for a ball tracking system". In: Fritz, M. et al. (Eds.). *ICVS*. Vol. 5815. Lecture Notes in Computer Science. Springer, pp. 435–444. ISBN: 978-3-642-04666-7.

Birbach, O. and U. Frese (2011). "Estimation and prediction of multiple flying balls using probability hypothesis density filtering". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2011)*, pp. 3426–3433. DOI: 10.1109/IROS.2011.6094622.

Birbach, O., J. Kurlbaum, T. Laue, and U. Frese (2008). "Tracking of ball trajectories with a free moving camera-inertial sensor". In: Iocchi, L. et al. (Eds.). *RoboCup*. Vol. 5399. Lecture Notes in Computer Science. Springer, pp. 49–60. ISBN: 978-3-642-02920-2.

Birbach, O., U. Frese, and B. Bäuml (2011). "Realtime perception for catching a flying ball with a mobile humanoid". In: *Proc. IEEE Intl. Conf. Robotics and Automation (ICRA2011), May 9-13, Shanghai, China*. IEEE, pp. 5955–5962.

Birchfield, S. (1998). "Elliptical head tracking using intensity gradients and color histograms". In: *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition, (CVPR 1998)*, pp. 232–237. DOI: 10.1109/CVPR.1998.698614.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. ISBN: 0387310738.

Blomdell, A., G. Bolmsjö, T. Brogårdh, P. Cederberg, M. Isaksson, R. Johansson, M. Haage, K. Nilsson, M. Olsson, T. Olsson, A. Robertsson, and J. Wang (2005). "Extending an industrial robot controller–

Implementation and applications of a fast open sensor interface". *IEEE Robotics & Automation Magazine* **12**:3, pp. 85–94.

Blomdell, A., I. Dressler, K. Nilsson, A. Robertsson, and I. Dressler (2010). "Flexible application development and high-performance motion control based on external sensing and reconfiguration of ABB industrial robot controllers". In: *Proc. ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications*. Anchorage, AK, pp. 62–66.

Borst, C., T. Wimböck, F. Schmidt, M. Fuchs, B. Brunner, F. Zacharias, P. Giordano, R. Konietschke, W. Sepp, S. Fuchs, C. Rink, A. Albu-Schäffer, and G. Hirzinger (2009). "Rollin' justin - mobile platform with variable base". In: *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pp. 1597–1598. DOI: 10.1109/ROBOT.2009.5152586.

Bouguet, J.-Y. (2010). *Camera calibration toolbox for matlab*. URL: http://www.vision.caltech.edu/bouguetj/calib_doc/.

Bruyninckx, H., T. Lefebvre, L. Mihaylova, E. Staffetti, J. De Schutter, and J. Xiao (2001). "A roadmap for autonomous robotic assembly". In: *Proc. Intl. Symp. Assembly and Task Planning*. Fukuoka, Japan, pp. 49–54.

Comaniciu, D. and P. Meer (1999). "Mean shift analysis and applications". In: *The Proceedings of the Seventh IEEE International Conference on Computer Vision, (ICCV 1999)*. Vol. 2, pp. 1197–1203. DOI: 10.1109/ICCV.1999.790416.

Dahl, O. and L. Nielsen (1989). "Torque limited path following by online trajectory time scaling". In: *Proceedings, IEEE International Conference onRobotics and Automation (ICRA'89)*, 1122–1128 vol.2. DOI: 10.1109/ROBOT.1989.100131.

Dai, S., M. Yang, Y. Wu, and A. K. Katsaggelos (2006). "Tracking motion-blurred targets in video". In: *Proc. Intl. Conf. Image Processing (ICIP2006)*. Atlanta, Georgia, USA, pp. 2389–2392.

De Schutter, J., T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx (2007). "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty". *Intl. J. Robotics Research* **26**:5, pp. 433–455.

Debrouwere, F., W. Van Loock, G. Pipeleers, Q. Tran Dinh, M. Diehl, J. De Schutter, and J. Swevers (2013). "Optimal robot path following for minimal time versus energy loss trade-off using sequential convex programming". In: *Proc. IEEE International Conference on Mechatronics*. Venice, Italy, pp. 316–320.

Di Lello, E., T. De Laet, and H. Bruyninckx (2012). "Hierarchical dirichlet process hidden markov models for abnormality detection in robotic assembly". In: *Workshop on Bayesian Nonparametric Models (BNPM) For Reliable Planning And Decision-Making Under Uncertainty (NIPS 2012)*. Lake Tahoe, USA.

Ding, T., M. Sznaier, and O. Camps (2007). "A rank minimization approach to fast dynamic event detection and track matching in video sequences". In: *46th IEEE Conference on Decision and Control*, pp. 4122–4127. DOI: 10.1109/CDC.2007.4434324.

Douxchamps, D. (2012). *libdc1394*. URL: http://damien.douxchamps.net/ieee1394/libdc1394/.

Du, H. and S. Nair (1999). "Modeling and compensation of low-velocity friction with bounds". *IEEE Trans. Control Systems Technology* **7**:1, pp. 110–121.

Ebner, M. (2007). *Color Constancy*. John Wiley & Sons, Chichester, England. ISBN: 9780470058299.

Einhorn, E., C. Schröter, H.-J. Böhme, and H.-M. Gross (2007). "A hybrid Kalman filter based algorithm for real-time visual obstacle detection". *Proc. 52nd Intl. Scientific Colloquium (IWK)* **II**, pp. 353–358.

Eom, K., I. Suh, W. Chung, and S. Oh (1998). "Disturbance observer based force control of robot manipulator without force sensor". In: *Proc. Intl. Conf. Robotics and Automation (ICRA1998)*. Leuven, Belgium, pp. 3012–3017.

Erkorkmaz, K. (2004). *Optimal trajectory generation and precision tracking control for multi-axis machines*. PhD thesis. Department of Mechanical Engineering, University of British Columbia.

Forsyth, D. A. and J. Ponce (2002). *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference. ISBN: 0130851981.

Fox, B. and K. Kempf (1985). "Opportunistic scheduling for robotic assembly". In: *Proceedings. IEEE International Conference on Robotics and Automation (ICRA'85)*. Vol. 2, pp. 880–889. DOI: 10.1109/ROBOT.1985.1087274.

Frese, U., B. Baeuml, G. Schreiber, I. Schaefer, M. Haehnle, G. Hirzinger, and S. Haidacher (2001). "Off-the-shelf vision for a robotic ball catcher". In: *Proc. IEEE/RSJ Intl. Conf. Intelligent Robots and Systems (IROS2001), October 2001, Maui*, pp. 1623–1629.

Geraerts, R. and M. H. Overmars (2002). "A comparative study of probabilistic roadmap planners". In: *Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR'02)*, pp. 43–57.

Graustein, W. C. (1930). *Homogeneous Cartesian Coordinates. Linear Dependence of Points and Lines. Ch. 3 in Introduction to Higher Geometry*. Macmillan, New York, pp. 29–49.

Hagander, P. (1973). *Operator Factorization and Other Aspects of the Analysis of Linear Systems*. PhD thesis TFRT-1005. Department of Automatic Control, Lund University, Sweden.

Harris, J. L. (1966). "Image evaluation and restoration". *J. Opt. Soc. Am.* **56**:5, pp. 569–570. DOI: 10.1364/JOSA.56.000569. URL: http://www.opticsinfobase.org/abstract.cfm?URI=josa-56-5-569.

Haschke, R., E. Weitnauer, and H. Ritter (2008). "On-line planning of time-optimal, jerk-limited trajectories". In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS2008)*, pp. 3248–3253. DOI: 10.1109/IROS.2008.4650924.

Hogan, N. (1985). "Impedance control: an approach to manipulation". *ASME J. Dynamic Systems, Measurement, and Control* **107**, pp. 1–24.

Hong, W. (1995). "Robotic catching and manipulation using active vision". In: *Master's thesis, Department of Mechanical Engineering, MIT*.

Hove, B. and J. Slotine (1991). "Experiments in robotic catching". In: *Proceedings of the 1991 American Control Conference (ACC'91)*. Vol. 1. Boston, MA, USA, pp. 380–385.

Hyland, J. (2002). "An iterated-extended Kalman filter algorithm for tracking surface and sub-surface targets". In: *OCEANS '02 MTS/IEEE*. Vol. 3. Biloxi, MS, USA, pp. 1283–1290.

JGrafchart (2013). URL: http://www.control.lth.se/Research/tools/grafchart.html.

Jia, J., J. Sun, C.-K. Tang, and H.-Y. Shum (2006). "Drag-and-drop pasting". In: *ACM Transactions on Graphics (SIGGRAPH 2006)*. Boston, Massachusetts, USA, pp. 631–636.

Jörg, S., J. Langwald, C. Natale, J. Stelter, and G. Hirzinger (2000). "Flexible robot-assembly using a multi-sensory approach". In: *IEEE Intl. Conf. Robotics and Automation (ICRA 2000)*. Vol. 4. San Francisco, CA, USA, pp. 3687–3694.

JR3 (2013). URL: http://www.jr3.com.

Kailath, T., A. Sayed, and B. Hassibi (2000). "Linear estimation". *Prentice Hall, Upper Saddle River, NJ*, pp. 310–361.

Kalman, R. (1960). "A new approach to linear filtering and prediction problems". *Transactions of the ASME–J. Basic Engineering* **82**:Series D, pp. 35–45.

Khatib, O. (1987). "A unified approach for motion and force control of robot manipulators: The operational space formulation". *IEEE J. Robotics and Automation* **3**:1, pp. 43–53.

Kober, J., M. Glisson, and M. Mistry (2012). "Playing catch and juggling with a humanoid robot". In: *Proc. IEEE-RAS Intl. Conf. Humanoid Robots*. Osaka, Japan.

Kock, S., T. Vittor, B. Matthias, H. Jerregård, M. Källman, I. Lundberg, R. Mellander, and M. Hedelind (2011). "Robot concept for scalable, flexible assembly automation: a technology study on a harmless dual-armed robot". In: *Proc. IEEE Int. Symp. Assembly and Manufacturing (ISAM2011)*. Tampere, Finland, pp. 1–5.

Kröger, T. and F. M. Wahl (2010). "On-line trajectory generation: Basic concepts for instantaneous reactions to unforeseen events". *IEEE Trans. on Robotics* **26**:1, pp. 94–111.

Kröger, T., A. Tomiczek, and F. M. Wahl (2006). "Towards on-line trajectory computation". In: *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2006)*. Beijing, China, pp. 736–741.

LabComm (2013). URL: http://wiki.cs.lth.se/moin/LabComm.

Lalonde, J.-F., D. Hoiem, A. A. Efros, C. Rother, J. Winn, and A. Criminisi (2007). "Photo clip art". *ACM Trans. Graph.* **26**:3. ISSN: 0730-0301. DOI: 10.1145/1276377.1276381. URL: http://doi.acm.org/10.1145/1276377.1276381.

Land, E. H. (1983). "Recent advances in retinex theory and some implications for cortical computations: color vision and the natural image". *Proc. National Acadmy of Sciences USA* **80**, pp. 5163–5169.

Lane, J. D. (1980). *Assembly Automation* **1**:1, pp. 36–46.

Lavalle, S. M. and J. J. Kuffner (2000). "Rapidly-exploring random trees: progress and prospects". In: Donald, B. R. et al. (Eds.). *Algorithmic and Computational Robotics: New Directions*. Wellesley, MA, pp. 293–308.

Levin, A., D. Lischinski, and Y. Weiss (2008). "A closed-form solution to natural image matting". *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **30**:2, pp. 228–242. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2007.1177.

Linderoth, M. (2008). *Vision Based Tracker for Dart Catching Robot*. Master's Thesis ISRN LUTFD2/TFRT--5830--SE. Department of Automatic Control, Lund University, Sweden. URL: http://www.control.lth.se/Publication/5830.html.

Linderoth, M. (2009). *Robot catching balls and darts*. URL: http://www.youtube.com/watch?v=Fxzh3pFr3Gs.

Linderoth, M. (2011). *Robotic Work-Space Sensing and Control*. Licentiate Thesis ISRN LUTFD2/TFRT--3251--SE. Department of Automatic Control, Lund University, Sweden. URL: http://www.control.lth.se/Publication/linderoth2011lic.html.

Linderoth, M., A. Robertsson, K. Åström, and R. Johansson (2010). "Object tracking with measurements from single or multiple cameras". In: *Proc. International Conference on Robotics and Automation (ICRA 2010)*. Anchorage, AK, USA, pp. 4525–4530.

Lippiello, V. and F. Ruggiero (2012a). "3D monocular robotic ball catching with an iterative trajectory estimation refinement". In: *Proc. IEEE Intl. Conf. Robotics and Automation (ICRA 2012)*. Saint Paul, Minnesota, USA, pp. 3950–3955.

Lippiello, V. and F. Ruggiero (2012b). "Monocular eye-in-hand robotic ball catching with parabolic motion estimation". In: *Preprints 10th IFAC Intl. Symp. Robot Control (SYROCO'12)*. Dubrovnik, Croatia.

Lotto, B. (2011). *Lotto Lab Studio, Illusions in colour perception*. URL: http://www.lottolab.org/articles/illusionsoflight.asp.

Lowe, D. G. (1999). "Object recognition from local scale-invariant features". In: *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2 (ICCV '99)*. Corfu, Greece, pp. 1150–1157. ISBN: 0-7695-0164-8. URL: http://dl.acm.org/citation.cfm?id=850924.851523.

Lu, J., E. Poon, and K. N. Plataniotis (2006). "Restoration of motion blurred images". In: *Proc. IEEE Intl. Conf. Multimedia and Expo (ICME 2006)*. Toronto, Ontario, Canada, pp. 1193–1196.

Lublinerman, R., M. Sznaier, and O. Camps (2006). "Dynamics based robust motion segmentation". In: *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition, (CVPR'06)*. Vol. 1. New York, NY, USA, pp. 1176–1184.

Ma, Y., S. Soatto, J. Kosecka, and S. S. Sastry (2003). *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer Verlag. ISBN: 0387008934.

Macfarlane, S. (2001). *On-Line Smooth Trajectory Planning for Manipulators*. M.A.Sc. thesis. Dept. Mech. Eng., Univ. British Columbia, Vancouver, Canada.

Maciejowski, J. (2002). *Predictive Control with Constraints*. Prentice Hall, Pearson Education, England.

Mason, M. T. (1981). "Compliance and force control for computer controlled manipulators". *IEEE Transactions on Systems, Man and Cybernetics* **11**:6, pp. 418–432. ISSN: 0018-9472. DOI: 10.1109/TSMC.1981.4308708.

Mattingley, J. and S. Boyd (2012). "CVXGEN: a code generator for embedded convex optimization". *Optimization and Engineering* **13**:1, pp. 1–27. URL: http://cvxgen.com/.

Möbius, F. (1827). *Die Barycentrische Calcül, Reprinted 1967 in Gesammelte Werke, vol. 1, pp. 36–49*. Dr. M. Saendig oHG, Wiesbaden, Germany.

Morrow, J., B. Nelson, and P. Khosla (1995). "Vision and force driven sensorimotor primitives for robotic assembly skills". In: *Proceedings. 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'95)*. Vol. 3, pp. 234–240. DOI: 10.1109/IROS.1995.525977.

Murakami, T., F. Yu, and K. Ohnishi (1993). "Torque Sensorless Control in Multidegree-of-Freedom Manipulator". *IEEE Trans. Industrial Electronics* **40**:2, pp. 259–265.

Nägele, F., M. Naumann, and A. Verl (2012). "A framework for a fault tolerant and learning robotic assembly system". In: *Proceedings of ROBOTIK 2012 - 7th German Conference on Robotics*. Munich, Germany, pp. 434–439.

Ohishi, K. (1993). "Sensorless force control using H^{∞}; acceleration controller". In: *Proc. Asia-Pacific Workshop on Advances in Motion Control*, pp. 13–18. DOI: 10.1109/APWAM.1993.316179.

Ohishi, K., M. Miyazaki, and M. Fujita (1992). "Hybrid control of force and position without force sensor". In: *Proc. Int. Conf. Industrial Electronics, Control, Instrumentation, and Automation, Power Electronics and Motion Control*. San Diego, USA, pp. 670–675.

Olsson, H., K. Åström, C. Canudas de Wit, M. Gäfvert, and P. Lischinsky (1998). "Friction models and friction compensation". *European Journal of Control* **4**:3, pp. 176–195.

Perez, P., C. Hue, J. Vermaak, and M. Gangnet (2002). "Color-based probabilistic tracking". In: *In Proc. European Conference on Computer Vision (ECCV 2002)*. Copenhagen, Denmark, pp. 661–675.

Popovic, M. and A. Goldenberger (1998). "Modeling of friction using spectral analysis". *IEEE Transactions on Robotics and Automation* **14**:1, pp. 114–122.

Raibert, M. H. and J. J. Craig (1981). "Hybrid position/force control of manipulators". *ASME Journal of Dynamic Systems, Measurement, and Control* **103**, pp. 126–133.

Real-Time Workshop (2011). URL: http://www.mathworks.com/products/simulink-coder/index.html.

Reid, D. (1979). "An algorithm for tracking multiple targets". *IEEE Transactions on Automatic Control* **24**:6, pp. 843–854. ISSN: 0018-9286. DOI: 10.1109/TAC.1979.1102177.

Riley, M. and C. Atkeson (2000). *Robot catching*. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.6087.

Rocco, P., G. Ferretti, and G. Magnani (1997). "Implicit Force Control for Industrial Robots in Contact with Stiff Surfaces". *Automatica* **33**:11, pp. 2041–2047.

Salari, V. and I. Sethi (1990). "Feature point correspondence in the presence of occlusion". *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12**:1, pp. 87–91. ISSN: 0162-8828. DOI: 10.1109/34.41387.

Schmidt, S. F. (1966). "Applications of state space methods to navigation problems". *Advanced Control Systems* **3**. Ed. by C. T. Leondes, pp. 293–340.

Schweitzer, H., J. W. Bell, and F. Wu (2002). "Very fast template matching". In: *Proceedings of the 7th European Conference on Computer Vision-Part IV (ECCV '02)*. Copenhagen, Denmark, pp. 358–372. ISBN: 3-540-43748-7. URL: http://dl.acm.org/citation.cfm?id=645318.649271.

Sezan, M. I. and A. M. Tekalp (1990). "Survey of recent developments in digital image restoration". *Optical Engineering* **29**:5, pp. 393–404.

Shen, C., J. Kim, and H. Wang (2010). "Generalized kernel-based visual tracking". *IEEE Trans. Circuits Syst. Video Techn.* **20**:1, pp. 119–130.

Simpson, J., C. Cook, and Z. Li (2002). "Sensorless force estimation for robots with friction". In: *Proc. Australasian Conf. Robotics and Automation (ACRA 2002)*. Auckland, New Zealand, pp. 94–99.

Soriano, M., B. Martinkauppi, S. Huovinen, and M. Laaksonen (2000). "Skin detection in video under changing illumination conditions." In: *Proc. 15th Intl. Conf. Pattern Recognition (ICPR 2000)*. Barcelona, Spain, pp. 839–842.

Spong, M. W., S. Hutchinson, and M. Vidyasagar (2006). *Robot Modeling and Control*. John Wiley & Sons, Hoboken, NJ. ISBN: 0471649902.

Stolt, A., M. Linderoth, A. Robertsson, and R. Johansson (2012). "Force controlled robotic assembly without a force sensor". In: *Proc. International Conference on Robotics and Automation (ICRA 2012)*. St. Paul, Minnesota, USA, pp. 1538–1543.

Stolt, A., M. Linderoth, A. Robertsson, and R. Johansson (2013). "Robotic assembly of emergency stop buttons". In: *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*. Tokyo, Japan.

Toshiba (2012). *Sensor-less compliance control for assembly robots*. URL: http://www.toshiba-machine.co.jp/en/technology/tech_catalog/e3.html.

Viola, P. and M. Jones (2001). "Robust real-time object detection". *International Journal of Computer Vision* **57**:2, pp. 137–154.

Wang, C., L.-S. Ho, and D. J. Cannon (1998). "Heuristics for assembly sequencing and relative magazine assignment for robotic assembly". *Computers & Industrial Engineering* **34**:2, pp. 423–431. ISSN: 0360-8352. DOI: http://dx.doi.org/10.1016/S0360-8352(97)00140-X. URL: http://www.sciencedirect.com/science/article/pii/S036083529700140X.

Weiner, L. B. (1981). "Kalman filter initialization with large initial uncertainty and strong measurement nonlinearity". *Proc. Region 3 Conf. and Exhibit, Huntsville, AL; United States*, pp. 150–151.

Wu, Y., J. Cheng, J. Wang, and H. Lu (2009). "Real-time visual tracking via incremental covariance tensor learning". In: *IEEE 12th International Conference on Computer Vision (ICCV 2009)*. Kyoto, Japan, pp. 1631–1638.

Wu, Y., H. Ling, J. Yu, F. Li, X. Mei, and E. Cheng (2011). "Blurred target tracking by blur-driven tracker". In: *Proceedings of the 2011 International Conference on Computer Vision (ICCV 2011)*. IEEE Computer Society, Washington, DC, USA, pp. 1100–1107. ISBN: 978-1-4577-1101-5. DOI: 10.1109/ICCV.2011.6126357. URL: http://dx.doi.org/10.1109/ICCV.2011.6126357.

Xenomai (2013). URL: http://www.xenomai.org.

Yilmaz, A., O. Javed, and M. Shah (2006). "Object tracking: a survey". *ACM Computing Surveys* **38**:4.

Yitzhaky, Y., R. Milberg, S. Yohaev, and N. S. Kopeika (1999). "Comparison of direct blind deconvolution methods for motion-blurred images". *Appl. Opt.* **38**:20, pp. 4325–4332. DOI: 10.1364/AO.38.004325. URL: http://ao.osa.org/abstract.cfm?URI=ao-38-20-4325.