Diploma thesis

Feedback control for planar parallel magnetic manipulation

Aram Simonian

Czech Technical University in Prague

Supervisor:
Ing. Jiří Zemánek

Faculty of Electrical Engineering
Department of Control Engineering
Czech Technical University in Prague

May 2014

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Aram Simonian**

Study programme: Cybernetics and Robotics
Specialisation: Systems and Control

Title of Diploma Thesis: **Feedback control for planar parallel magnetic manipulation**

Guidelines:

The goal of this thesis is to design and implement a feedback control system for planar manipulation with several objects in a magnetic field created by a planar array of coils. The control system should be able to find and/or improve a suitable control strategy. Adaptive control, reinforcement learning etc. can be used.

1. Finish the image processing system for measuring of the objects' positions.
2. Design a learning control system for parallel manipulation.
3. Implement and test the control system on both PC platform and Speedgoat real-time rapid prototyping platform.

Bibliography/Sources:

[1] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.
[2] F. L. Lewis, D. Liu, Reinforcement Learning and Approximate Dynamic Programming for Feedback Control, Wiley, 2012.
[3] K. F. Böhringer, H. Choset, Ed., Distributed Manipulation, Springer, 2000.

Diploma Thesis Supervisor: Ing. Jiří Zemánek

Valid until the winter semester 2014/2015

[illegible]

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
4
L.S.

VZ. [illegible]

prof. Ing. Michael Šebek, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, September 16, 2013

Abstract

This thesis deals with self-learning feedback control of a platform for planar non-contact magnetic manipulation. First, the work briefly describes the instrumentation of the experimental setup used for evaluation of suggested control methods. Second, it involves a computer vision part for object detection that is used for image-based feedback control. The main body of the work resides in study of different adaptive and learning methods that allow either to develop and enhance an existing controller or to find a suitable control strategy from scratch in a trial-and-error manner. Finally, the suggested methods are implemented in MATLAB & Simulink and tested on the real magnetic manipulator using the PC platform and Speedgoat real-time rapid prototyping platform.

Abstrakt

Tato diplomová práce se zabývá zpětnovazebním učícím se řízením platformy pro bezkontaktní magnetickou manipulaci. Řešenou problematiku lze rozdělit do dvou částí. První část spočívá v návrhu systému počítačového vidění pro detekci polohy manipulovaných objektů v obraze z kamery snímající povrch platformy. Detekovaná poloha slouží pro uzavření obrazové zpětné vazby. Druhá část práce prozkoumává možnosti učících se adaptivních metod, které umožňují buď vylepšovat existující řízení, nebo nalézt vhodnou řídicí strategii bez předchozí znalosti řízeného systému. Navržené metody jsou implementovány v prostředí MATLAB & Simulink a otestovány na reálném modelu ve spojení s platformou PC i s real-time platformou Speedgoat.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 11. 5. 2014

[illegible]

podpis

Acknowledgement

Foremost, I would like to thank to Ing. Jiří Zemánek for his leadership, improving ideas and valuable advice. Many thanks go also to Subramanya Nageshrao and Prof. Dr. Robert Babuška from TU Delft for helping me to get a better insight in reinforcement learning and to Dr. Zdeněk Hurák and Kristian Hengster-Movric, Ph.D. for interesting and inspirative discussions. I would also like to express great gratitude to my family for their support, patience and calm study environment they provided.

Contents

1. Introduction

Adaptive and learning control methods are becoming more and more popular these days, because of the need to accomplish more complex control tasks and reach higher efficiency of the resulting control process. They allow us not only to react adequately to the changes of parameters of the controlled system in real-time or enhance an existing control, but also to solve difficult control problems that we would not be able to solve using traditional control design methods at all ([1], [46], [27]). Modern control systems must be able to extract and exploit the experience from the control process on-the-go, typically to behave optimally with respect to given criteria. These criteria can vary quite a lot, from the best possible tracking of reference signal, to the minimum-energy control to time-optimal control and others. In this thesis we attempt to apply adaptive and learning control methods to a magnetic platform for planar manipulation shown in Figure 2.1a, which is designed to manipulate one or several steel balls on its surface.

1.1. Outline

The introductory Chapter 2 describes the magnetic platform, its instrumentation and the mathematical model that is used throughout the thesis. The rest of the thesis is organized in two main thematic blocks, which are the computer vision part (Chapter 3) and the control part (Chapters 4, 5, 6 and 7). Chapter 3 is dedicated to the computer vision system for position measurement of the manipulated objects on the surface of the platform. Beside the description of the detection algorithms, the section also presents the achieved experimental results in ball detection to preserve continuity. Chapter 4 addresses the theoretical aspects of reinforcement learning and its application to control of dynamic systems. The following Chapter 5 presents an adaptive control scheme for differently sized balls. Finally, the Chapters 6 and 7 describe the results achieved with the presented control methods both in simulations and with the real platform.

1.2. Motivation and scope

The task of steering a steel ball using an array of electromagnets is an instance of a broader class of problems, which is the manipulation by shaping force fields. Therefore the magnetic manipulator (Magman) platform can be regarded as a benchmark system for testing advanced control algorithms for a whole class of problems. The device has a spatially distributed set of actuators that allow shaping the force field in order to control position and/or orientation of a manipulated object. Similar manipulation devices are referred to as programmable force fields or programmable vector fields in the control community ([7], [9], [32]). However, the authors address devices manipulating larger objects with a MEMS[1] based array of micro-actuators, that have ability to produce unit force along a single dimension. According to [8] the particular micro-actuators implementation can be based on various principles including the vibratory surface, electrostatic field, magnetic field, thermo-bimorph cilia or pressurized air nozzles.

The spatially discrete micro-actuators combined with large manipulated object allow quite fine shaping of the resulting force field with respect to the object. By contrast, Magman's actuators are comparable in size or even bigger than the manipulated object and produce a continuous force field, which implies higher requirements on the control of the actuators. In addition, the magnetic actuators cannot produce repelling force. They can only attract a ferromagnetic object, which results in a more constrained control problem.

The force field manipulation offers several advantages compared to the standard contact manipulation using some kind of gripper. First, it can be realized as contactless, which allows manipulating micro-scale objects that could not be grasped by any gripper. Second, different force fields, for example electric or magnetic, can spread throughout various environments that prevent contact manipulation, e.g. aggressive chemicals or the interior of human body. In addition, the force fields also allow massive parallelization, manipulating a whole swarm of objects at once, not focusing on every single object separately.

Overall, the force field manipulation can open new application areas. On the other hand, the applications are quite challenging, because it is difficult to precisely model the force fields for the subsequent controller design. Precise modeling of a magnetic field produced by an array of coils with iron cores is a particularly complicated problem. In addition, the exerted magnetic field is influenced by the magnetizable manipulated object, which complicates the situation even more. The motivation for this thesis is to explore the possibilities of learning and adaptive control methods, that may be able to overcome problems caused by simplified modeling of the magnetic field and the manipulated object. The methods can use some prior knowledge about the system represented by the simplified model and improve the model with the control strategy according to the experienced behavior of the system, or they can even learn some control strategy from scratch, without using any prior knowledge at all.

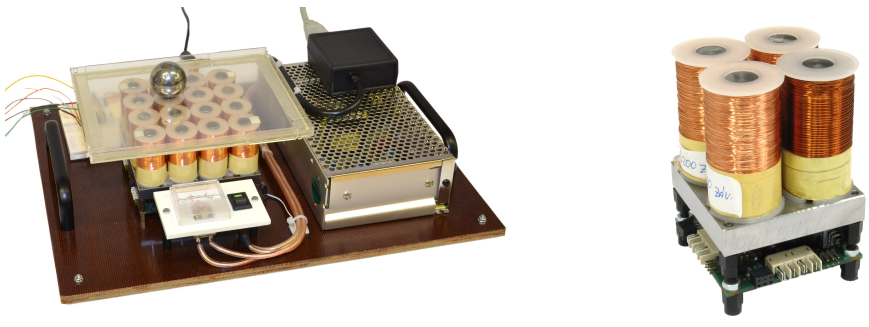
[1]Microelectromechanical systems

2. Magnetic platform description

The Magman magnetic platform used throughout this work is a laboratory model developed in the AA4CC group. The platform consists of a modular array of coils that allow steering a ferromagnetic ball on the platform's surface by shaping the magnetic field produced by the coils. The ball is rolling on the surface of the platform at all times, so there is no dragging or even levitation. The coil array is currently composed from four independent modules such as the one in Figure 2.1b, each module carrying four coils with iron cores connected to an iron slab. The modules allow independent PWM current control for each of their coils. There is a board with electronics for coil current control, communication and an ARM processor beneath the coils, which means that the modules can not only communicate with its neighbors, but they also have their own computing power which can be used for distributed control algorithms in the future. From the control systems point of view, the platform is a distributed multiple-input multiple-output nonlinear dynamic system. In this thesis, however, we will only consider centralized control algorithms with central sensing element and distributed actuation. In fact, it is the distributed and highly nonlinear character of the actuators which makes any control task quite challenging.

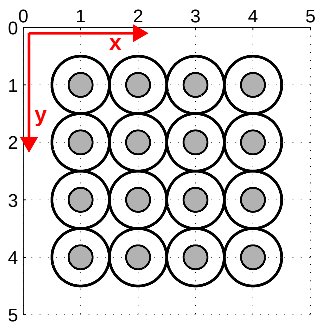
2.1. Magman mathematical model

This section briefly describes the mathematical model of a ball rolling on the surface of the platform, controlled by a coil array with a feedback linearization of its input nonlinearity developed in [49]. Both the model and the feedback linearization concept are used throughout this thesis. The model is not completely precise and neglects

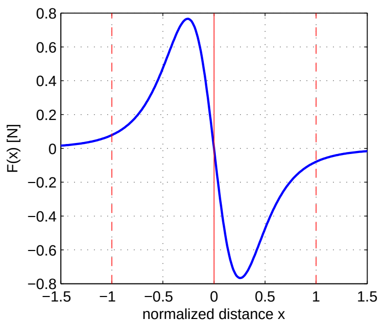


(a) Magman platform **(b)** a single module

Figure 2.1. The magnetic manipulator platform (a) is composed from four independent modules. A detail of a single module is shown in Figure (b). Photographs by J. Zemánek.



(a) top view of the platform



(b) 1-D force profile of a single coil

Figure 2.2. Top view of the platform and force profile for a single coil. The scheme of the top view in (a) shows the orientation of axes for normalized platform coordinates. The Figure (b) shows the identified one-dimensional profile of magnetic force exerted by a single fully energized coil on a 30 mm ball. The solid red line marks the center of the activated coil, while the dashed red lines mark the centers of the neighboring coils.

certain phenomena, which is the motivation for application of adaptive and learning control. The basic idea is to use the advanced control methods to eliminate the influence of model's imprecision on the resulting performance of the controlled system.

The rolling of the ball is decoupled to the x- and y-component, modeling the movement in each direction independently as a double integrator

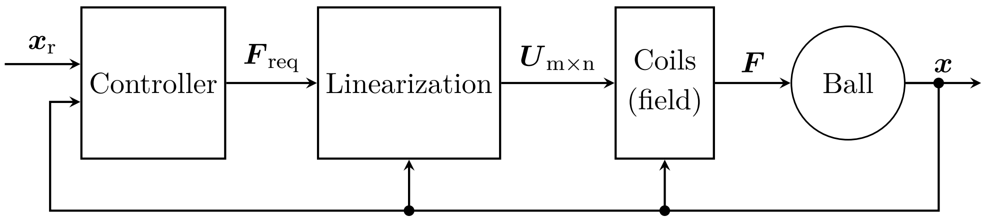
$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \hdashline \dot{y} \\ \dot{v}_y \end{bmatrix} = \left[\begin{array}{cc:cc} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array}\right] \begin{bmatrix} x \\ v_x \\ \hdashline y \\ v_y \end{bmatrix} + \left[\begin{array}{c:c} 0 & 0 \\ m_{\text{ef}}^{-1} & 0 \\ \hdashline 0 & 0 \\ 0 & m_{\text{ef}}^{-1} \end{array}\right] \begin{bmatrix} F_x \\ \hdashline F_y \end{bmatrix} \tag{2.1}$$

where the constant m_{ef} is the effective mass of the ball, which merges its rotational and translational inertia and F_x, F_y are the components of magnetic force. The integrator is undamped, so the model neglects the rolling resistance between the rolling ball and the surface of the platform and the eddy currents and hysteresis of the induced magnetic field in the body of the ball. The position and velocity of the ball are expressed in normalized platform units, where one unit corresponds to the distance of centers of two adjacent coils, i.e. 25 mm in reality (see Figure 2.2a).

The components of the magnetic force are determined by superposition of contributions of individual coils of the array as

$$\begin{aligned} F_x &= \sum_{m=1}^{4}\sum_{n=1}^{4} U_{m,n} \frac{-a(x-m)}{\left((x-m)^2+(y-n)^2+b\right)^3} \qquad (2.2) \\ F_y &= \sum_{m=1}^{4}\sum_{n=1}^{4} U_{m,n} \frac{-a(y-n)}{\left((x-m)^2+(y-n)^2+b\right)^3}, \qquad a,b = const. > 0 \end{aligned}$$

where the fractional expression is an analytical model of magnetic force produced by a single fully energized coil at position $[m,n]$ and $U_{m,n}$ is the corresponding coil activation factor, which scales the exerted magnetic force linearly. The scaling is realized



$\boldsymbol{x}_\mathrm{r}$
Controller
$\boldsymbol{F}_\mathrm{req}$
Linearization
$\boldsymbol{U}_{\mathrm{m}\times\mathrm{n}}$
Coils (field)
$\boldsymbol{F}$
Ball
$\boldsymbol{x}$

Figure 2.3. Scheme of the feedback linearization approach. The linearization block solves an optimization problem to find the coil activation factors that result in an approximation of the magnetic force requested by the controller.

by PWM control of the coil current. The pure superposition of contributions of individual coils is a simplification, which has been justified experimentally by magnetic field measurements.

Taking advantage of the radial symmetry of the magnetic field produced by a single coil, the model of the magnetic force was experimentally identified as one-dimensional force profile, i.e. the magnetic force as a function of only one coordinate x. During the experiment, a 30 mm ball was pushed by a force gauge above an energized coil back and forth, simultaneously recording the position of the ball. The measured data set was fitted with an analytical expression

$$F(x) = -\frac{ax}{(x^2+b)^3} \quad [\mathrm{N}], \qquad a = 0.1857, \;\; b = 0.3303 \tag{2.3}$$

where $x = 0$ corresponds to the coil center and the constants a, b are the same as in Equations 2.2. The identified force profile for the 30 mm ball presented in Figure 2.2b shows that the effect of an individual coil is spatially limited and strongly depends on the position of the ball.

2.1.1. Feedback linearization

The input nonlinearity of the coil array is solved by a feedback linearization approach developed in [26] that uses numerical optimization to compute the coil activation factors, which after superposition of coil contributions result in exerting a requested magnetic force in the x- and y-direction. After reordering the matrix of coil activation factors into a vector and denoting the fractional expressions from Equations 2.2 as $G_{xm,n}(x,y)$ and $G_{ym,n}(x,y)$, the overall force is expressed as

$$\underbrace{\begin{bmatrix} G_{x1,1}(x,y) & \dots & G_{x4,4}(x,y) \\ G_{y1,1}(x,y) & \dots & G_{y4,4}(x,y) \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} U_{1,1} \\ \vdots \\ U_{4,4} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} F_x \\ F_y \end{bmatrix}}_{\mathbf{b}}. \tag{2.4}$$

The system of equations 2.4 is underdetermined, so a criterion has to be chosen for the solution. A 2-norm of the coil activation factor vector was chosen to avoid intense activation of the coils. Because the array cannot exert an arbitrarily large force in a given direction, the criterion was extended with the 2-norm of the exerted force error term to allow an approximate solution in the cases that do not allow an exact solution. With this extension, the coil activation factors are determined by solving the problem

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & ||\mathbf{x}|| + c||\mathbf{A}\mathbf{x} - \mathbf{b}|| \\ \text{subject to} \quad & 0 \leq \mathbf{x} \leq 1 \end{aligned} \tag{2.5}$$

where the constant $c > 0$ has to be set high enough to ensure negligible force error in the situations when it is possible to obtain an exact solution. Removing the input nonlinearity of the coil array allows using a commanded magnetic force for the ball position control, as shown in Figure 2.3. The feedback linearization is used as an underlying layer for the reinforcement learning in this thesis and is also adopted for the auto-identification adaptive scheme described in Chapter 5. After mastering the control with the use of the feedback linearization, the reinforcement learning is planned to be used to directly control the activation factors of the coils.

2.2. Instrumentation

This section provides a brief description of the additional important devices and specific software used for experiments and their mutual interconnection.

2.2.1. Speedgoat rapid prototyping platform

Speedgoat is a Swiss company which specializes in manufacturing devices for real-time simulations, which are specifically designed to be used as external targets with the xPC Target software. The external target workflow proceeds as follows:

- First, a Simulink model has to be created on a so-called host PC, which is an ordinary PC with MATLAB, MATLAB Coder, Simulink, Simulink Coder and xPC Target installed.
- Second, the model is compiled into an executable file and uploaded to the external target, which is connected to the host PC.
- Third, the execution of the compiled model on the target is launched from the host PC, allowing online parameter tuning and data import and export. With an extra license for xPC Target Embedded Option, it is even possible to create fully standalone target applications that can be deployed and run without connection to the host PC.

The Speedgoat Performance real-time machine that we use for our experiments is physically an ordinary PC based on Intel chipset with several PCI I/O expansion cards including both analog and digital programmable inputs and outputs, an I/O FPGA board, RS-232/422/485 serial ports, 2-port CAN module and BitFlow NEON-CLB framegrabber for a Camera Link compatible camera. The presence of the Camera Link framegrabber card was reflected when choosing a suitable camera, which is described in the following section. Beside Speedgoat, other platforms were considered, too, including the CompactRIO from National Instruments or Raspberry Pi. Speedgoat was preferred to CompactRIO because it can be targeted directly from

MATLAB/Simulink and we were already familiar with this environment, while the CompactRIO uses LabVIEW, which would be new for us. Compared to Raspberry Pi, Speedgoat offers a wider range of interfaces (CAN, Camera Link, etc.), which makes it more versatile for other applications than the magnetic platform control in the future.

The external target model execution is analogous to the “External mode” model execution with Real-Time Windows Target but it should in general offer even higher computing performance. The external target runs xPC Target kernel during the execution instead of a more complex operating system like Windows, so it can effectively schedule hardware resources and guarantee real-time response, without the risk that the simulation will be slowed down by another process. However, the use of an external target also poses some constraints on the executed Simulink model. Because of the compilation, all blocks in the model have to support C code generation, which is typically not fulfilled for some more complex blocks e.g. from Image Processing Toolbox.

We faced some unexpected limitations of xPC Target in the area of Camera Link support. There are two ways how to grab video from a Camera Link camera interfaced over Bitflow Neon framegrabber in Simulink. The first option is through the block From Video Device, which is a part of Image Acquisition Toolbox. The block works fine in the normal mode in Simulink, but it does not support code generation, so it cannot be used in the external mode. The other option is the BitFlow CameraLink block from the xPC Target library, which supports the code generation, but it does not allow hardware Bayer interpolation[1], which is essential for us, because it saves a lot of computation time. To get the hardware Bayer interpolation support, MathWorks would have to release an update of the framegrabber’s driver, which would make use of its full functionality.

The xPC Target has been updated and renamed to Simulink Real-Time recently. However, MathWorks only allows access to product’s technical documentation to the users that have purchased Simulink Real-Time license so we cannot comment on possible improvements in Camera Link support and video processing. There appeared quite useful guidelines for Speedgoat target machine setup at the manufacturer’s website [21], which were not available at the time when we were starting using it and which would have been very helpful. Some practical tips originating from our experience can be found in Appendix B.

2.2.2. Camera

The other device we would like to briefly describe is the fast RGB camera Basler acA2000–340kc, which is used for ball position measurement, i.e. providing visual feedback. It is a high speed industrial camera based on a CMOS image sensor with resolution 2046×1086 pixels. The camera connects to a framegrabber over Camera

[1]Our color camera provides raw data from its sensor, so we first get a single-layer image, where 50%of pixels represent green, 25% blue and 25%red component of the image and the data has to be interpolated to get a standard three-layer RGB image.

Link interface, which is besides Gigabit Ethernet and USB 3.0 another common standard for high speed devices. The camera can reach up to 340 frames per second at full resolution in the extended full Camera Link configuration. However, we want to process all the camera data in real-time, so we use it in base Camera Link configuration below 100 frames per second. It would not be possible to process the data at camera's full frame rate online without a specialized hardware (FPGA or GPU computation). Although Speedgoat machine has an FPGA module, it is intended as a reconfigurable 64-channel I/O interface, thus it is not suitable for processing the data from a camera. If necessary, the base Camera Link configuration allows to reach frame rate up to 112 frames per second. The base Camera Link configuration also allows using more affordable framegrabbers. We use the BitFlow NEON-CLB framegrabber, which comes with MATLAB adapter software and is partially supported by xPC Target library in Simulink. There occurred several technical issues during the camera and framegrabber setup, which are addressed in Appendix A.

2.2.3. Ball position measurement

At the time we started working on this thesis, a working ball position measurement solution was already implemented. This original solution uses a resistive touch foil, whose analog output data is acquired by Humusoft MF624 data acquisition card connected to a PC. The main advantage of the solution is that it can achieve quite high sampling frequency in order of kilohertz. Another interesting feature is the possibility to estimate the pressure of the ball on the surface of the resistive foil from the resulting resistance of the foil. The main limitation of the resistive foil is that it can only measure position of a single object[2], which arises from the physical principle of the measurement of object's position. In addition, the foil needs a sufficient pressure on its surface to detect the position of the measured object, which becomes a problem when one wants to perform experiments with a smaller ball that is not heavy enough to produce the needed pressure.

For the parallel manipulation, it is necessary to measure position of several objects simultaneously, so we decided to design a computer vision system for ball detection, monitoring the platform from the top view with a camera. Unlike the resistive foil, the computer vision system is not limited only to a single object only and therefore can be used for feedback control of several objects at once. Furthermore, the weight of the ball does not cause problem either and we also expect that the visual position measurement will be less noisy than the one using the touch foil. On the other hand, we expect that it will not be possible to reach as high sampling frequency as with the resistive foil, because the object detection in a video-sequence is more computationally demanding than processing the analog outputs of the resistive touch foil.

[2]There are also multi-touch resistive foils available on the market (e.g. from Stantum company), but the platform is only equipped with a single-touch one. The multi-touch foils have the surface divided into separate domains and every single domain can detect position of one object at a time.

2.3. Speedgoat–Magman interface

For the experiments, we need to connect the magnetic platform to Speedgoat machine. The camera cannot be used with Speedgoat due to the limited Camera Link support in xPC Target mentioned in Section 2.2.1, so we use the resistive foil for the experiments with a single ball and Speedgoat machine. The hardware interconnection with the magnetic platform is quite straightforward thanks to the variety of I/O interfaces offered by Speedgoat. We use the digital and analog inputs and outputs of the Speedgoat module IO101 to acquire the ball position data from the resistive foil. The serial communication with the platform is transmitted over the Speedgoat serial port module IO503 configured in the RS-485 mode.

Even though we use the same signal processing methods as on the PC platform, the position measured using the Speedgoat interface is remarkably more noisy than with the Humusoft MF624 card on the PC. The standard deviation of measured position on Speedgoat is $\sigma_x = 0.045$ in the normalized platform units, which is three times higher compared to the standard deviation $\sigma_x = 0.015$ achieved on the PC platform. The increased noise of the measurement is caused by the Magman's power supply, which is used for powering the electronics that allow the readout of the touch foil data when connecting to Speedgoat. When we use a stabilized laboratory power supply instead, the standard deviation of the measurement on Speedgoat drops to $\sigma_x = 0.004$, which is even below the value on the PC. Unfortunately, we did not manage to identify the cause of the higher noise before finishing the control experiments, so we might have been able to reach better results with the Speedgoat platform than the ones that will be presented here. The repetition of experiments remains for future work due to the time reasons.

3. Computer vision

In this chapter, we describe different algorithms for detection of position of one or more steel balls in the video-sequence obtained from camera, which monitors the magnetic platform from the top view. Our aim is to obtain a real-time detection method that would give a precision of detected position good enough for applicability of the image-based feedback. The higher frequency and precision we achieve, the better control we can get. Our first estimate of a reasonable target sampling frequency of the computer vision system is 40 Hz. The estimate originates from empirical experience with the feedback control of a ball using the resistive touch foil measurement. The motivation for the image-based feedback is given by the possibility to measure position of several balls at once, not being limited by the size of the balls.

3.1. Problem setting

In our detection problem, we have a scene with constant illumination and distance from the camera, which is guaranteed to be static except for a detected object or objects. The detected objects have known visual appearance and they can be either moving or located statically anywhere in the scene.

The algorithms will be described for a single object to preserve simplicity and clarity. The algorithms can be afterwards extended for multiple objects by simply executing the operations for every single object every iteration, except for the background subtraction method described in Section 3.2, which is only suitable for single object detection and is described here for completeness of documentation of work. Our application is simplified by the fact that the detected objects cannot occlude each other.

As the image data comes from a real experiment and we are sampling a continuous movement of the detected object, we know that the magnitude of difference of object's positions in consecutive video frames is limited. This allows us to make use of the inter-frame information. We do not have to search for the object in the whole scene every time, but we can search in a restricted region of interest (ROI) around the previously detected position instead to optimize speed. In the cases when the previous position of the ball is unknown, the ROI is set to the whole image.

Because of the need for real-time detection, it is not possible to apply complex detection algorithms requiring a lot of computation. When only propagating the captured video frames to Simulink workspace without any further processing, the system reaches maximum frame rate 60 Hz, which is only 20 Hz above the target value mentioned in the beginning of this chapter, not leaving a lot of computation time for computer vision algorithms. The camera and framegrabber would allow a higher sampling frequency, as already mentioned in Section 2.2.2, but the bottleneck of the video acquisition chain is in the propagation of the data from the framegrabber card to Simulink workspace.

Although the early experiments were conducted with a steel ball without any coating, we finally decided to paint the manipulated balls with color for several reasons. First, it will be useful for the parallel manipulation, where the color distinguishes different balls. Second, a well-marked color of a ball can simplify its detection and third, using a matte paint can partially suppress reflections of surrounding at the ball's surface, which cause problems during the ball detection otherwise.

There are many different detection and tracking methods available in the computer vision area. Majority of them focuses on more complex and structured objects than we need (e.g. cars, pedestrians, faces). Besides that, the computer vision is not the problem we want to focus on the most. Even though there are some overviews and surveys available such as [48] or [35], it is quite demanding to orientate oneself in such variety. We have chosen a few simple methods that will be described in this chapter and afterwards implemented for experiments.

3.1.1. Notation for computer vision

We will use lowercase bold Roman letters such as $\boldsymbol{x}$ to denote vectors. Uppercase bold Roman letters such as $\boldsymbol{X}$ will denote matrices, while the elements of matrices will be referred to as X_{ij} or $\boldsymbol{X}(i,j)$. When referring to the size of matrices or images as well as to coordinates of an element of a matrix or image, we will preserve the usual convention $x \times y$ for size and (x, y) for coordinates, where x denotes the horizontal size or column index and y denotes the vertical size or row index. The origin of the system of coordinates is located in the upper left corner of the matrix or image, with x-axis pointing to the right and y-axis pointing downwards.

3.2. Background subtraction

Different methods for foreground object detection based on background subtraction are described in [33]. The methods rely on maintaining a model of the image background, which allows us to separate the foreground, i.e. the moving objects in the scene. When the model of the background is incrementally updated by new data, it is possible to separate the foreground even from a varying background under the condition that the background changes are slow compared to the changes in foreground. According to [33], the running Gaussian average, temporal median filter and mixture of Gaussians should provide the best computational speed (in the same order).

In our application, we have the advantage of a completely static background of the scene, so we can build a model of the background in advance and use it for foreground detection without updating afterwards. The model of the background is represented as a matrix of the mean pixel intensity values of the field of view (FOV) of the camera. Omitting the incremental update of the scene model speeds up the computation compared to the detection methods mentioned above, but on the other hand, it makes the algorithm more sensitive to change of illumination, for example. However, we can afford losing robustness in exchange for the higher computation speed, because we can ensure constant illumination of the scene.

3.2.1. Initialization

The model of background is measured before placing the detected ball into the field of view (FOV) of the camera. We use sample mean value computed from N consecutive frames as a model of the background. We have also tried using temporal median value instead sample mean value, because median filtering is less sensitive to values biased with extremal additive error than averaging. However, unlike median filtering, averaging can be implemented iteratively as in (3.1), without storing all the data acquired over time in memory.

$$\boldsymbol{B}_{i+1} = \frac{1}{i+1}\left(i\boldsymbol{B}_i + \boldsymbol{I}_i\right), \qquad \boldsymbol{B}_0 = \mathbf{0}, \quad i = 0 \dots N, \tag{3.1}$$

where $\boldsymbol{B}$ is the estimate of the mean background at the i-th iteration, $\boldsymbol{I_i}$ is the i-th acquired video frame and N is the total number of video frames used for estimation of the background. The number of frames used for mean background computation was chosen with respect to the standard deviation of pixel values in time caused by the additive image noise, which was estimated experimentally. The image noise for every pixel has approximately Gaussian distribution with zero mean and standard deviation 4.7 (measured on the 0-255 intensity value scale). Having Gaussian distribution of the image noise, we can claim that the standard deviation σ_M of sample mean value computed from N samples is

$$\sigma_M = \frac{1}{\sqrt{N}}\sigma_x, \tag{3.2}$$

where σ_x is the standard deviation of the original random variable. With $\sigma_x = 4.7$ and $N = 400$, we achieve standard deviation of pixel values of background model

$$\sigma_B = 0.235. \tag{3.3}$$

The value σ_B is a measure of the uncertainty of the background model. In this case, it tells us that 95% of samples of a single image pixel will fall to the particular bin of 0-255 scale, which corresponds to the mean value given by the model.

3.2.2. Detection

Having the model of the mean background, we can proceed to detection. The position of the ball is detected using the absolute difference of the image $\boldsymbol{I}$ captured in the current time step and the model of the static background $\boldsymbol{B}$. The detection process has the following phases:

1. compute difference image $\boldsymbol{D}$ from captured image $\boldsymbol{I}$ and mean background model $\boldsymbol{B}$,

2. classify each pixel of the difference image as background, foreground, or “not determined”,
3. estimate the position of the center of the ball from classifications of the difference image.

The difference image $\boldsymbol{D}$ computed in phase 1 is a two-dimensional matrix of the same size as the size of the region of interest. The pixel values of difference image $\boldsymbol{D}$ are computed as Euclidean distance in RGB color space of every pixel of ROI in captured image $\boldsymbol{I}$ from the corresponding part of the mean background model $\boldsymbol{B}$

$$\boldsymbol{D}_{\mathrm{M}\times\mathrm{N}} : \{\boldsymbol{D}(i,j) = ||\boldsymbol{I}_{\mathrm{ROI}}(i,j) - \boldsymbol{B}_{\mathrm{ROI}}(i,j)||_{\mathrm{RGB}}\}\,. \tag{3.4}$$

In phase 2, we use two thresholds t_1, t_2 to classify the pixels of difference image into three classes

$$\mathcal{C}(D_{ij}) = \begin{cases} 0 & \text{iff} \quad 0 < D_{ij} \leq \mathrm{t}_1, \\ 1 & \text{iff} \quad \mathrm{t}_1 < D_{ij} \leq \mathrm{t}_2, \\ 2 & \text{otherwise}, \qquad\qquad \mathrm{t}_1 < \mathrm{t}_2. \end{cases} \tag{3.5}$$

Classification $\mathcal{C}(D_{ij}) = 0$ denotes assignment of the pixel to the background, $\mathcal{C}(D_{ij}) = 2$ denotes foreground (the ball) and finally $\mathcal{C}(D_{ij}) = 1$ represents the “not determined” class. The higher threshold t_2 is chosen empirically as a quantile of the difference image pixel values. Therefore it is set to such value that n pixel values from the difference image were above t_2. The quantity n is one half of the amount of pixels that would belong to the ball in the image if the classification was ideal. There is a protective lower bound for t_2. The bound prevents setting too low t_2 in situations when the ball is lost, which would lead to random behavior of the detection algorithm. The lower threshold t_1 is chosen as one half of the threshold t_2.

In phase 3, we spatially filter the classifications of difference image pixels from phase 2. First, we form two vectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, whose lengths are equal to the width M and height N of the difference image, respectively, so that $\boldsymbol{v}_1$ contains sums of columns and $\boldsymbol{v}_2$ contains sums of rows of the difference image

$$\begin{aligned} \boldsymbol{v}_1 &= \sum_{j=1}^{N} \mathcal{C}(D_{ij}), \\ \boldsymbol{v}_2 &= \sum_{i=1}^{M} \mathcal{C}(D_{ij}). \end{aligned} \tag{3.6}$$

Second, we form a digital filter $\boldsymbol{f}_c$ whose coefficients are given as lengths of secant lines of a circle, all perpendicular to the same axis, as shown in Figure 3.1. Motivation for these specific filter coefficients comes from computation of vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. Given an ideal-case classification of ideal difference image, vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ would contain lengths of secant lines of a circle with diameter of the ball in pixels, with some offset given by the size of background. Expression for $\boldsymbol{f}_c$ as a function of distance d from the center of the filter can be obtained from analytical equation that characterizes circle in plane, centered at the origin with diameter r, which can be understood as implicit

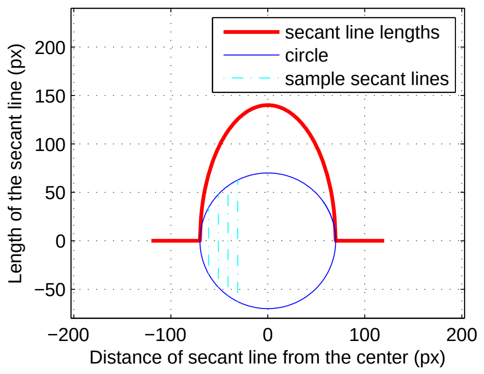


Figure 3.1. Length of parallel secant lines of a circle with radius 70 px as a function of their distance from the circle's center (red line), i.e. the coefficients of filter f_c. The blue circle represents projection of a ball to the image plane. A few sample secant lines are shown as well, plotted with cyan color.

function of x. Considering only the upper half of the circle above the horizontal axis, we remove ambiguity and get lengths of upper halves of the secant lines

$$y = \sqrt{r^2 - x^2}, \qquad x \in \langle -r; r \rangle \tag{3.7}$$

which yields filter coefficients $f_c(x)$

$$f_c(x) = \begin{cases} 2\sqrt{r^2 - x^2} & \text{iff} \quad x \in \langle -r; r \rangle, \\ 0 & \text{otherwise.} \end{cases} \tag{3.8}$$

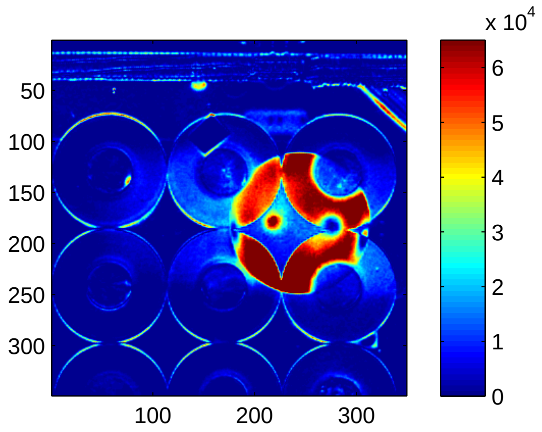
Finally, we estimate the coordinates x_c, y_c of the center of the ball within the region of interest by finding maximal values in vectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ filtered with the digital filter $\boldsymbol{f}_c$

$$\begin{aligned} x_c &= \arg\max_i \left\{ (\boldsymbol{v}_1 * \boldsymbol{f}_c)(i) \right\}, \\ y_c &= \arg\max_j \left\{ (\boldsymbol{v}_2 * \boldsymbol{f}_c)(j) \right\}, \end{aligned} \tag{3.9}$$

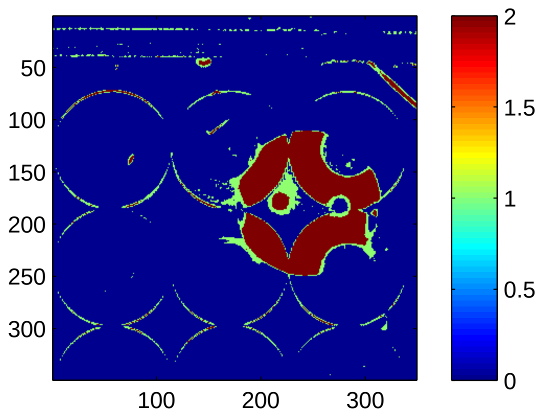
where we are indexing the central parts of the discrete convolutions, which have the same length as original vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. Estimated coordinates of the center of the ball in the whole image are afterwards obtained from x_c, y_c by adding the offset of the upper left corner of ROI. The whole detection process with background subtraction is illustrated in Figure 3.2.

3.3. Meanshift

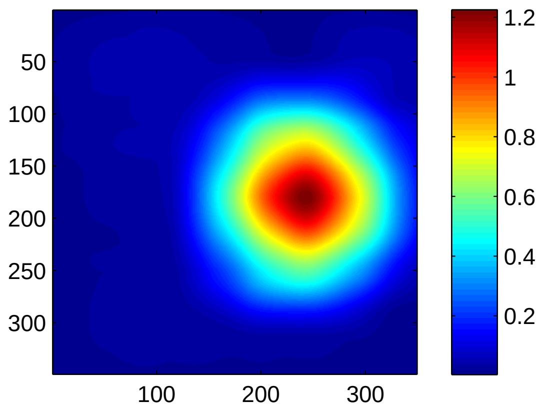
Meanshift is a fast and effective nonparametric method for moving object tracking in video sequences. It is a simplified variant of camshift algorithm described in [10] or [3]. Camshift is an abbreviation for "Continuously Adaptive Mean Shift". As its name suggests, the camshift algorithm can adapt to changes of the visual appearance



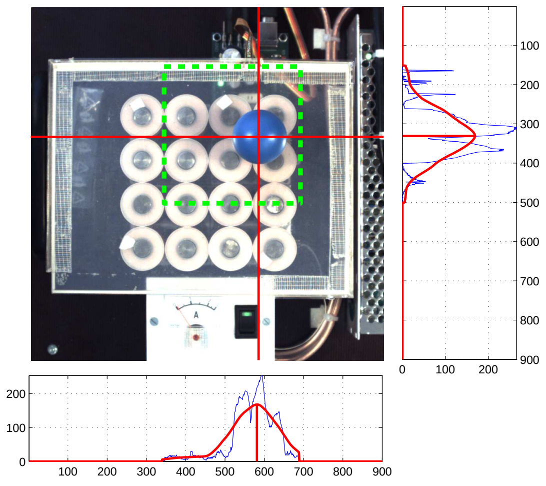
(a) ROI of the difference image



(b) ROI segmentation after thresholding



(c) spatial filtering of ROI segmentation



(d) original image with ROI and detection

Figure 3.2. Detection process with background subtraction algorithm. Image (a) shows ROI of difference image in terms of Euclidean distance in the RGB space. Image (b) shows segmentation of the ROI into background and foreground after thresholding. Image (c) is only intended to illustrate the effect of spatial filtration on the segmented ROI. It is not computed during detection. Image (d) shows the original input image with detected ball position and ROI in context with the row and column sums and their filtered values.

of the tracked object caused for example by partial occlusions, as well as to changes of scale of the object in the image caused e.g. by movement of the object, which significantly changes the distance of the object from the camera. Furthermore, it can adapt to rotation of the object, too. In our application, the object has constant visual appearance and the scale changes due to the movement of the object in the field of view of the camera are negligible. In addition, the ball is radially symmetrical, so we do not need adaptation to rotation either. These circumstances allow us to use the meanshift algorithm described in [29] and [36].

The meanshift algorithm searches for such image region, whose pixel intensity value histogram fits the reference histogram of the detected object the best. Therefore it uses prior knowledge of the visual appearance of the ball represented by histograms of RGB or HSV pixel intensity values. The RGB color space is often used in the image processing, but as mentioned in [48], it is not a perceptually uniform color space, i.e. the distance of colors in the color space is not always proportional to the difference of respective colors perceived by human eye. In addition, there is some correlation between the values in R, G and B channels (e.g. change of intensity of illumination of the scene affects the intensities of all channels in a similar way), which indicates that HSV color space histograms can be more discriminative in some situations for the detection purposes than the RGB ones. On the other hand, cameras provide image signal in RGB representation, so HSV histograms require further computation to transform the color spaces. Qualitative comparison of meanshift applied to different color spaces is given in Section 3.6.

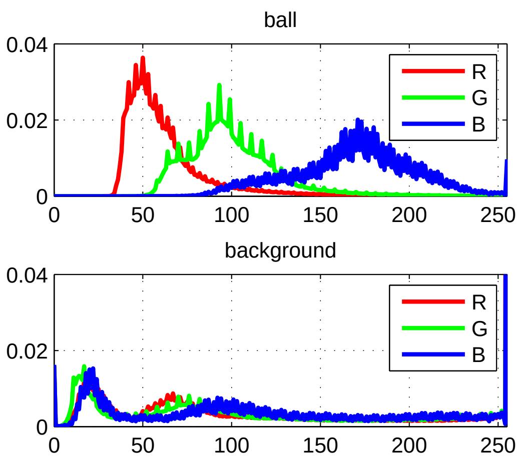
For general description of the algorithm, it does not make any difference whether we use RGB or HSV color space image representation, so we assume throughout the description that every pixel $\boldsymbol{I}(i, j)$ of the acquired image represents an RGB triplet of values $0-255$, supposing that the situation for the HSV case is analogous in the sense that each image pixel is described by a triplet of numbers.

3.3.1. Initialization

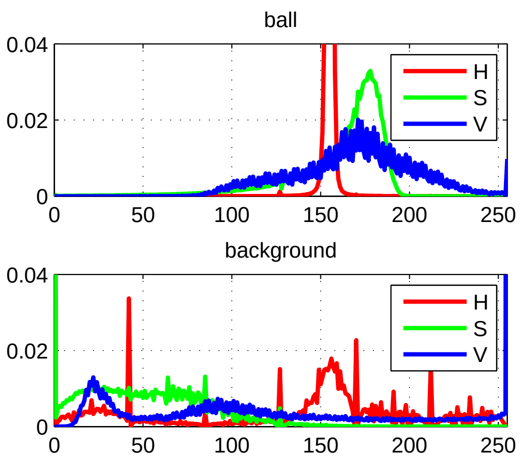
Before we start the detection, we collect and store the prior knowledge about the problem in a suitable representation. The prior knowledge about visual appearance of the ball and the scene is represented by histograms of pixel intensity values in the desired color space and is extracted from manually annotated video frames. The frame annotation contains information about the circular envelope of the ball in the image and can be used for extraction of the prior knowledge as well as for evaluation of precision of ball detection.

The histograms extracted from the annotated video frames approximate conditional probability mass functions of occurrence of pixel intensity values (coordinates in the RGB color space), given the information whether it belongs to the ball or to the background. We collect six histograms in total, three histograms for the ball and another three for the background, every histogram representing a single color channel. The histograms are denoted as h_a^C, where C denotes channel of the image (R, G, or B in the RGB case) and a represents the condition whether the histogram relates to the background ($a = 0$) or to the foreground ($a = 1$). Each of the histograms has 256 bins corresponding to the integral pixel intensity values in range from 0 to 255 and is normalized so that it sums to one to approximate the probability mass functions of occurrence of a particular intensity value in the foreground and in the background respectively. Figure 3.3 shows histograms of intensity values extracted from manually annotated video frames with a blue ball. The upper graphs contain histograms for the pixels belonging to the ball, while the lower ones contain histograms for the background.

Even though we have already mentioned that the RGB channels are correlated in general, we will consider them being mutually independent. The correlation of the



(a) pixel intensity histograms in RGB space



(b) pixel intensity histograms in HSV space

Figure 3.3. Normalized histograms of pixel intensity values of a blue ball (the upper ones) and background (the lower ones) for both discussed color spaces. The histograms are normalized by the total number of pixels included in the statistic to show relative numerosity of particular pixel intensity value.

RGB channels is caused by the fact that a change of intensity of illumination affects all the channels in the same way (higher resp. lower illumination intensity increases resp. decreases values in all RGB channels). Under a constant illumination of the scene, we neglect the channel correlation, which allows us to compute following conditional probabilities for a pixel $\boldsymbol{I}(i,j)$ characterized by RGB triplet $\boldsymbol{x} = \begin{bmatrix} x_\mathrm{R} & x_\mathrm{G} & x_\mathrm{B} \end{bmatrix}^T$

$$P(\boldsymbol{x}|0) = P(x_\mathrm{R}|0)P(x_\mathrm{G}|0)P(x_\mathrm{B}|0) \approx h_0^\mathrm{R}(x_\mathrm{R})h_0^\mathrm{G}(x_\mathrm{G})h_0^\mathrm{B}(x_\mathrm{B}), \tag{3.10}$$

$$P(\boldsymbol{x}|1) = P(x_\mathrm{R}|1)P(x_\mathrm{G}|1)P(x_\mathrm{B}|1) \approx h_1^\mathrm{R}(x_\mathrm{R})h_1^\mathrm{G}(x_\mathrm{G})h_1^\mathrm{B}(x_\mathrm{B}). \tag{3.11}$$

Term $P(\boldsymbol{x}|0)$ denotes the probability of occurrence of the particular triplet $\boldsymbol{x}$ given the information that the corresponding pixel belongs to the background, while $P(\boldsymbol{x}|1)$ denotes the probability of occurrence of triplet $\boldsymbol{x}$ given the fact that it belongs to the ball, i.e. the foreground.

3.3.2. Detection

During the detection process, we use the prior knowledge about the visual appearance of the ball to detect its position in the image. The detection proceeds in three consecutive steps

1. backprojection of conditional probabilities into the original image,
2. maximum likelihood classification of image pixels,
3. iterative computation of the probabilistic center of mass of pixels classified as foreground.

In the phase 1, the histograms of pixel intensity values collected during the initialization process are used to obtain two probability images $\boldsymbol{P}_0$, $\boldsymbol{P}_1$, which are two-dimensional matrices of the same size as the size of ROI in pixels. The pixel values of image $\boldsymbol{P}_0$ are computed according to equation 3.10, while the pixel values of $\boldsymbol{P}_1$ are determined using 3.11.

During the classification phase 2, every pixel of the ROI is assigned either to the background or to the foreground using maximum likelihood estimation. Maximum likelihood method can be used to determine parameters of a model from observed data and prior knowledge. Our model of a pixel has only one binary parameter c that acts as a "foreground indicator", with value $c = 1$ denoting the foreground. We are maximizing the likelihood function for every pixel of the ROI separately

$$\mathcal{L}(c|\boldsymbol{x}) = P(\boldsymbol{x}|c) \approx h_c^{\mathrm{R}}(x_{\mathrm{R}})h_c^{\mathrm{G}}(x_{\mathrm{G}})h_c^{\mathrm{B}}(x_{\mathrm{B}}), \qquad c \in \{0; 1\}. \tag{3.12}$$

An image pixel I_{ij} characterized by the particular RGB triplet $\boldsymbol{x}$ is therefore classified as

$$\mathcal{C}(\boldsymbol{x}) = \arg\max_c \left\{\mathcal{L}(c|\boldsymbol{x})\right\}. \tag{3.13}$$

Thus the maximum likelihood estimation of parameter c is equivalent to pixel-wise comparison of probability images $\boldsymbol{P}_0$ and $\boldsymbol{P}_1$. The classification phase yields a binary image $\boldsymbol{B}$, which is used as a mask for the final detection phase.

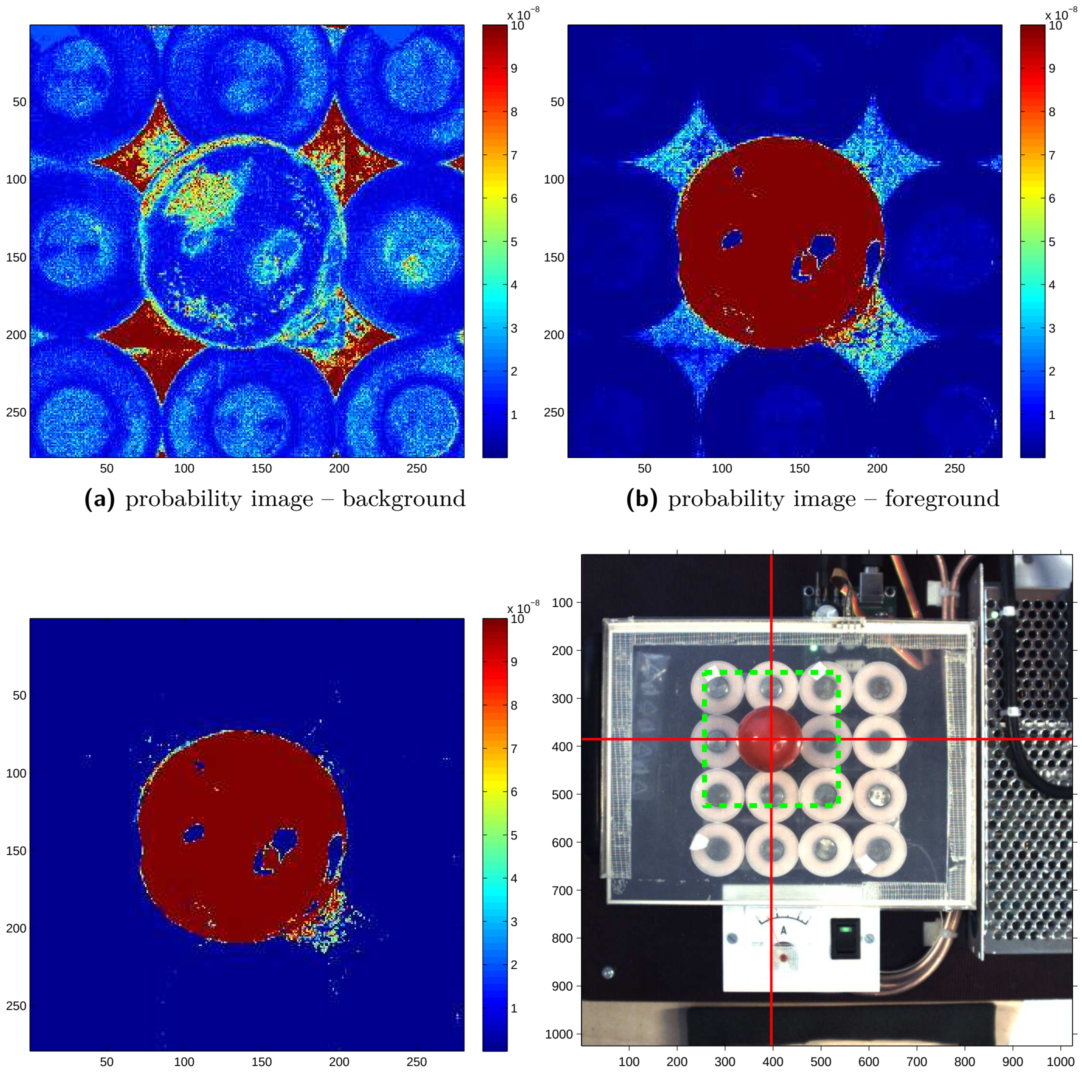
In the final phase 3, the algorithm iteratively computes the coordinates of the center of the ball as the center of mass of foreground pixels in the ROI, where the foreground pixels are weighted by their likelihood function value $\mathcal{L}(1|\boldsymbol{x})$. The coordinates x_c, y_c of the center of mass of foreground pixels are determined as

$$\begin{aligned} x_c &= \frac{1}{k}\sum_{i=1}^{M}\left[i\sum_{j=1}^{N}\mathcal{C}(\boldsymbol{x}_{ij})\mathcal{L}(\mathcal{C}(\boldsymbol{x}_{ij})|\boldsymbol{x}_{ij})\right], \\ y_c &= \frac{1}{k}\sum_{j=1}^{N}\left[j\sum_{i=1}^{M}\mathcal{C}(\boldsymbol{x}_{ij})\mathcal{L}(\mathcal{C}(\boldsymbol{x}_{ij})|\boldsymbol{x}_{ij})\right], \\ k &= \sum_{i=1}^{M}\sum_{j=1}^{N}\mathcal{C}(\boldsymbol{x}_{ij})\mathcal{L}(\mathcal{C}(\boldsymbol{x}_{ij})|\boldsymbol{x}_{ij}), \end{aligned} \tag{3.14}$$

where the normalization constant k also acts as an indicator of quality of detection, M is width of ROI in pixels and N is height of ROI in pixels. For the next iteration, the ROI is centered at x_c, y_c and a new position of center of mass is determined, which repeats until the position converges. In practice, we assume that the position has converged when the distance of mass centers from two consecutive iterations is shorter than a small positive constant. After convergence, x_c and y_c are equal to the coordinates of the estimated position of the center of the ball.

3.3.3. RGB vs. HSV color space

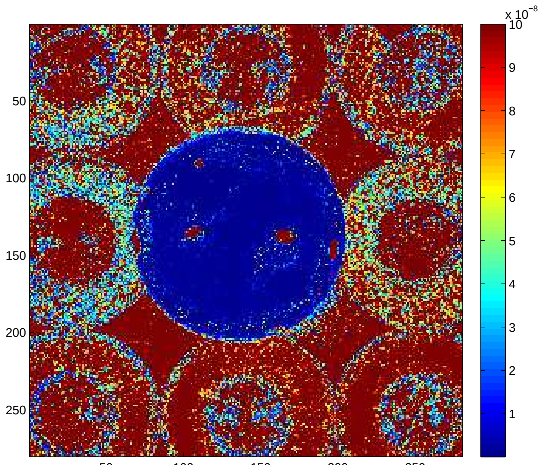
In this section we briefly compare the RGB and HSV variants of meanshift algorithm described in previous section. Comparing the histograms of the pixel intensity values shown in Figures 3.3a and 3.3b, it is noticeable that there is a more remarkable difference between the background and foreground histograms in the HSV case. Especially

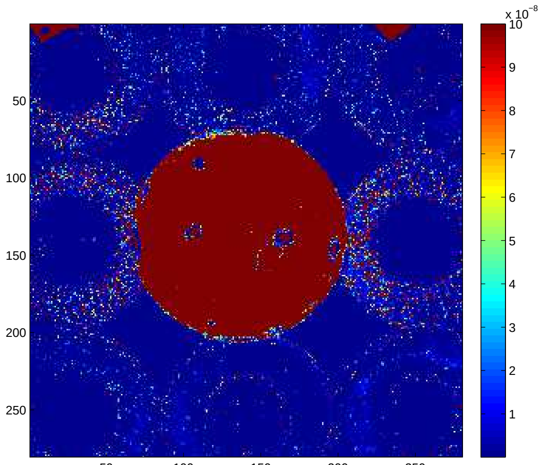


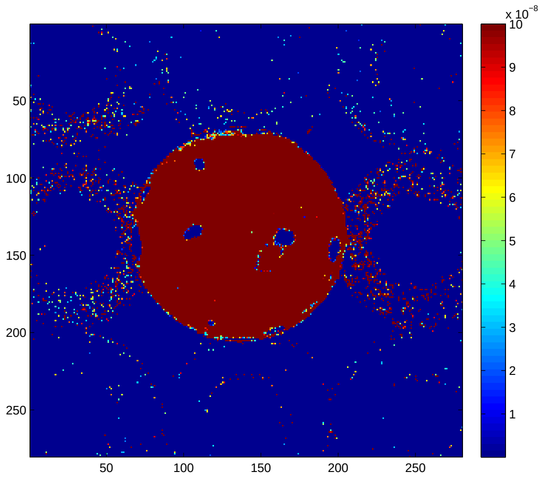
(a) probability image – background **(b)** probability image – foreground

(c) foreground pixel weights **(d)** original image with ROI and detection

Figure 3.4. Detection process with meanshift algorithm in the RGB color space. Image (a) shows backprojection of background likelihood into the ROI. Image (b) shows backprojection of foreground likelihood into the ROI. Image (c) shows weights of the ROI pixels after the maximum likelihood classification. Image (d) shows the original input image with the detected ball position and ROI.


(a) probability image – background


(b) probability image – foreground


(c) foreground pixel weights

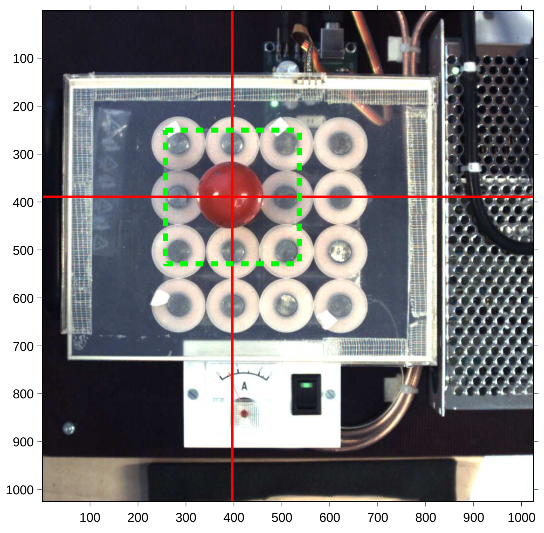

(d) original image with ROI and detection

Figure 3.5. Detection process with meanshift algorithm in the HSV color space. Image (a) shows backprojection of background likelihood into the ROI. Image (b) shows backprojection of foreground likelihood into the ROI. Image (c) shows weights of the ROI pixels after the maximum likelihood classification. Image (d) shows the original input image with the detected ball position and ROI.

the *hue* channel is quite discriminative. We have to be aware that the histograms can be visually misleading, because they show relative numerosity of intensity values for separate channels, not in the context of pixels, as it is utilized during likelihood estimation 3.12. The separability of background and foreground pixels using intensity values would be better visualized as a scatter plot in an orthogonal 3D space, whose dimensions would be interpreted as color space channels.

When we compare the detection processes in RGB and HSV color space illustrated in figures 3.4 and 3.5, we can see that the HSV case fits reality better than the RGB one, especially during the background likelihood backprojection (3.4a and 3.5a), where the RGB histograms do not work well. Despite this difference, the resulting quality of detection is comparable. A more detailed analysis of speed and precision of all presented methods is given in Section 3.6.

3.4. Linear SVM pixel classification

Although the meanshift algorithm described in the previous section works well, it is not fast enough for our purposes. The most computationally expensive part of its detection process is the likelihood backprojection into the ROI pixels. We were therefore looking for a different method that would allow us to distinguish between background and foreground pixels, which led us to SVM. Support vector machines, also called support vector networks [15], are a popular group of classifiers from supervised learning methods. There exist several variants of SVM [13] including linear, non-linear, two-class, multi-class or soft margin SVM.

From the classification point of view, our feature space is the RGB color space and we have two classes of pixels, the background and the foreground. As the background and foreground are not strictly linearly separable in the RGB color space, we decided to choose the soft margin variant of two-class linear SVM like the one analyzed in [19]. The general idea of two-class linear SVM is to construct a boundary hyperplane in the feature space, which separates the two classes and maximizes the distance of the closest learning samples from the boundary. The data points closest to the boundary hyperplane are then called support vectors.

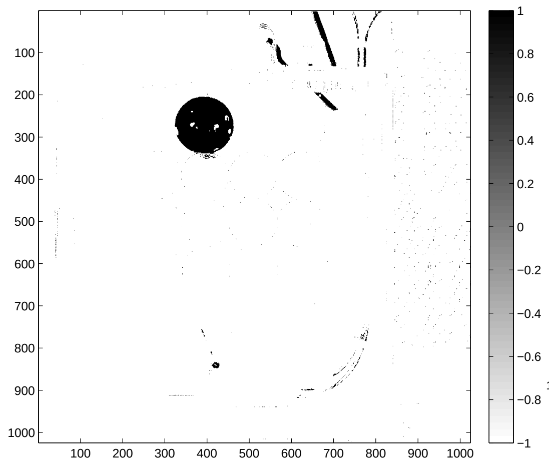
A linear SVM classifier is parametrized by a vector of weights $\boldsymbol{w}$ and a bias b. When these are known, the classification of a sample characterized by feature vector $\boldsymbol{x}$ into one of two classes proceeds as

$$\mathcal{C}(\boldsymbol{x}) = \frac{1}{2}\left[1 + \operatorname{sgn}\left(\boldsymbol{w}\boldsymbol{x} + b\right)\right]. \tag{3.15}$$

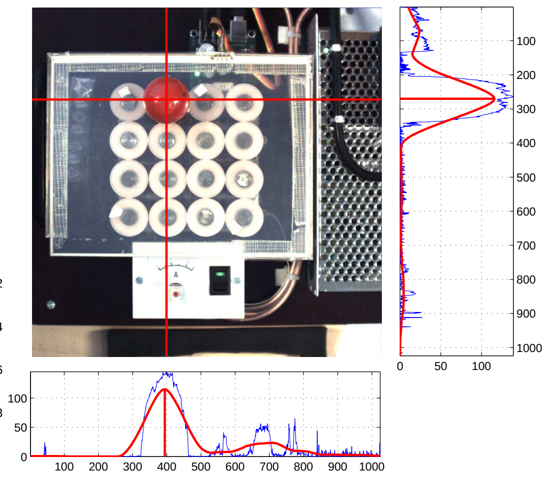
To learn the weights and bias of the classifier, we use a set of labeled training samples. The learning process is equivalent to an optimization task

$$\min_{\boldsymbol{w},b} \{||\boldsymbol{w}||_2\} \qquad \text{s.t.} \qquad \forall i:\ y_i(\boldsymbol{w}\boldsymbol{x}_i + b) \geq 1, \tag{3.16}$$

where $y_i \in -1, 1$ is the label of training sample $\boldsymbol{x}_i$. However, the optimization task 3.16 is unfeasible for problems that are not linearly separable. Therefore the



(a) SVM pixel classification



(b) original image with detection

Figure 3.6. Detection process with SVM pixel classification in the RGB color space. Image (a) shows result of SVM pixel classification with foreground pixels black and background pixels white. Image (b) shows the original input image with the detected ball position in context with the row and column class sums and their filtered values. ROI was not used when generating the figures to show the false foreground pixels that would otherwise lay outside of the ROI.

soft margin SVM introduces for every training sample $\boldsymbol{x}_i$ a slack variable s_i, which allows misclassifications in the training data. The slack variables change the optimization task 3.16 to

$$\min_{\boldsymbol{w},b,\boldsymbol{s}} \{||\boldsymbol{w}||_2 + \xi||\boldsymbol{s}||_2\} \text{ s.t. } \forall i: \; y_i(\boldsymbol{w}\boldsymbol{x}_i + b) \geq (1 - s_i), \; s_i \geq 0, \tag{3.17}$$

where the slack variable weight ξ is a tuning parameter which specifies how important it is to avoid misclassifications. In addition, we can use a vector of slack variable weights instead of the scalar value ξ, specifying slack variable weight for each training sample separately. This is particularly useful when we need to eliminate misclassifications of samples from one class at the cost of more misclassifications of the elements from the other class. Overall, the slack variables bring more degrees of freedom, which allows to apply SVM to linearly non-separable problems, too, but on the other hand, it also increases the number of optimized parameters, which leads to higher computational requirements and destroys scalability for large training data sets. Nevertheless, neither the number of parameters of learnt SVM classifier nor the classification speed are affected by the slack variables.

Both optimization tasks 3.16 and 3.17 can be effectively implemented using a quadratic programming solver, e.g. the *quadprog* function from MATLAB's Optimization toolbox.

3.4.1. Detection

We are not able to eliminate all false foreground pixels in the SVM classification even when using different slack variable weights for background and foreground training samples. It is not surprising, because we are using color as a feature and the background contains objects with color similar to the ball. We can eliminate the influence of the false foreground pixels by the same spatial filtering of pixel classifications as used in phase 3 of the background subtraction method described in Section 3.2.

The detection process with the SVM classifier is illustrated in Figure 3.6. We can see that the SVM classifier output depicted in Figure 3.6a gives much better estimation of foreground pixels than the thresholding of distance from mean background shown in Figure 3.2b, which results in a more precise detection of the ball's position. The resulting detection of ball position in Figure 3.6b is quite precise (see Table 3.2 in Section 3.6), despite the numerous false foreground pixels visible in Figure 3.6a. Moreover, unlike the background subtraction, the linear SVM method can simultaneously detect more than one ball, using different SVM classifiers for different balls.

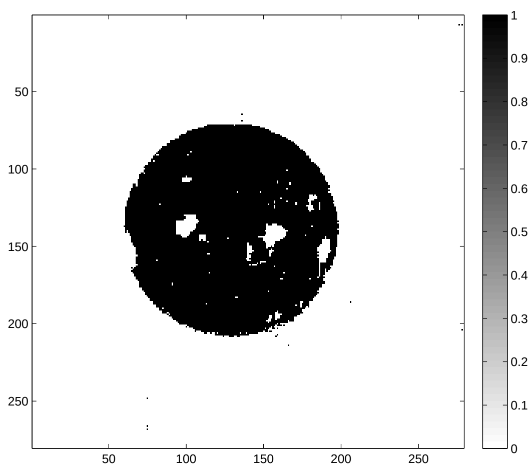
3.5. HSV feature thresholding

The last detection method we present here is the HSV feature thresholding. The method is inspired by high discriminativity of the *hue* channel in histograms shown in Figure 3.3b. The idea is to set lower and upper bound on the *hue* (H), *saturation* (S) and *value* (V) channels to distinguish between the background and foreground pixels.

The bounds for HSV channels are set using histograms of HSV values of ball pixels extracted from the annotated video frames. For every channel, we start with equal lower and upper bound located at the maximum of the histogram. Afterwards, we decrease the lower bound and increase the upper one until the range between the bounds covers 95% of ball pixels in the given channel. These bounds are then used for pixel classification. Every pixel, whose H, S and V features lay within the defined ranges, is classified as foreground, all the others are considered as background:

$$\mathcal{C}(\boldsymbol{x}) = \begin{cases} 1 & \text{iff} \quad H_{\text{low}} \leq x_{\text{H}} \leq H_{\text{upp}} \quad \wedge \quad S_{\text{low}} \leq x_{\text{S}} \leq S_{\text{upp}} \quad \wedge \\ & \qquad V_{\text{low}} \leq x_{\text{V}} \leq V_{\text{upp}}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.18}$$

According to our empirical experience, it is possible to omit the upper bound on *saturation* and both bounds on *value* for successful ball detection, which saves some computation time. The classification 3.18 results in a binary image which is further processed by the spatial filtering which was also used in phase 3 of the background subtraction method (see Section 3.2). The detection process with HSV thresholding is illustrated in Figure 3.7.


(a) binary ROI after HSV thresholding

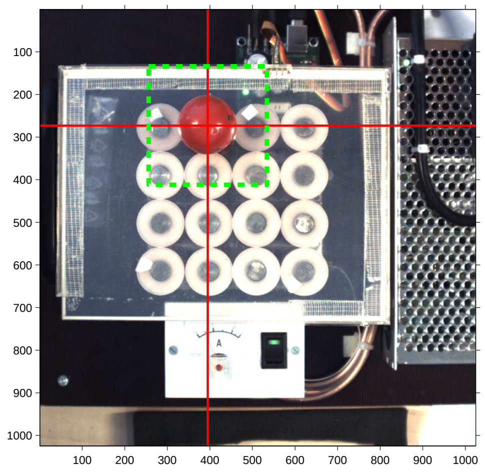

(b) original image with detection

Figure 3.7. Detection process with thresholding in the HSV color space. Image (a) shows the binary image resulting from the thresholding process. Image (b) shows the original input image with the detected ball position and ROI marked with green dashed line

3.5.1. HSV thresholding in the RGB color space

While the HSV feature thresholding is computationally cheap, the transformation of RGB camera image to the HSV color space turned to be more lengthy than the HSV thresholding itself. Even though there is a non-linear transformation from RGB to HSV color space, we found out that it is possible to perform the HSV thresholding directly in the RGB color space, without any further transformation. When we look at the hue, saturation and value isosurfaces in the RGB space (see Figure 3.8), it is obvious that they are linear or piecewise linear in terms of RGB components. If we describe the isosurfaces corresponding to the hue, saturation and value bounds analytically using plane equations, we will be able to do the HSV thresholding by simply checking whether a pixel lays above or below a set of boundary planes in the RGB space.

The *hue* isosurfaces shown in Figure 3.8a can be represented as boundary planes defined by an oriented line segment $\vec{\boldsymbol{l}}$, which is common for all hue values, and a specific point $\boldsymbol{A}$, whose location depends on the particular value of hue. The oriented line segment is the space diagonal of the RGB cube, which starts at the origin of the RGB color space and ends at point $\boldsymbol{Q} = \begin{bmatrix} 255 & 255 & 255 \end{bmatrix}^T$. The base location of point $\boldsymbol{A}$ for hue level $H = 0$ is

$$\boldsymbol{A} = \begin{bmatrix} 255 & 0 & 0 \end{bmatrix}^T.$$

The locations of point $\boldsymbol{A}$ for other values of hue are generated by rotating the base location of point $\boldsymbol{A}$ around $\vec{\boldsymbol{l}}$ in positive direction about angle

$$\varphi = 2\pi H_x, \qquad H_x \in \langle 0; 1),$$

where H_x is the hue value we are interested in.

The *saturation* isosurfaces shown in Figure 3.8b have a more complicated structure than the ones for hue. Such surface can be described as a hexagonal pyramid with apex located at the origin of the RGB color space and axis pointing in the direction of line segment $\vec{\boldsymbol{l}}$ which was defined above for the hue boundary planes. The base vertices of the pyramid are located at the edges and face diagonals of the RGB cube that pass through point $\boldsymbol{Q}$, which was also defined for the hue boundary planes. The location of the vertices on the edges and diagonals depends on the particular value of saturation. In the singular case, when $S = 0$, all of the vertices overlay at $\boldsymbol{Q}$. When the saturation increases, the vertices move along the corresponding edges and diagonals, so that their distance d_i from point $\boldsymbol{Q}$ is

$$d_i = \bar{d}_i S_x, \qquad S_x \in \langle 0; 1),$$

where $\bar{d}$ is the length of the corresponding edge or face diagonal and S_x is the saturation value.

The *value* isosurfaces in Figure 3.8c are structured, too. Every of them consists of three mutually perpendicular planes defined by three neighboring faces of a cube with one vertex located in the origin of the RGB color space and edges parallel with the edges of the RGB cube. The length of edges of the cube that defines the boundary planes is linear with respect to the value of value channel.

$$a = 255 V_x, \qquad V_x \in \langle 0; 1),$$

where 255 is edge length of the RGB cube and V_x is the particular value of value channel.

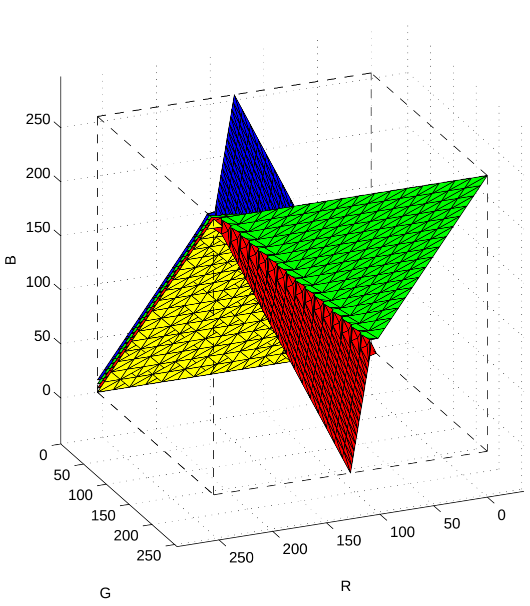
After computing parametrization of boundary planes for particular HSV limits, only the minimal required subset of them is selected for thresholding. A typical selection for ball detection is illustrated in Figure 3.8d, which depicts the situation for lower and upper hue bound and a lower saturation bound. The other bounds were omitted. The red saturation boundary planes in Figure 3.8d were rejected for the classification purpose, because they wouldn't affect the classification due to the selected hue bounds.

HSV thresholding in the RGB color space brings a great computational speed improvement compared to the thresholding of an image converted to HSV color space. According to statistics presented in Table 3.1, the computation speed nearly tripled.

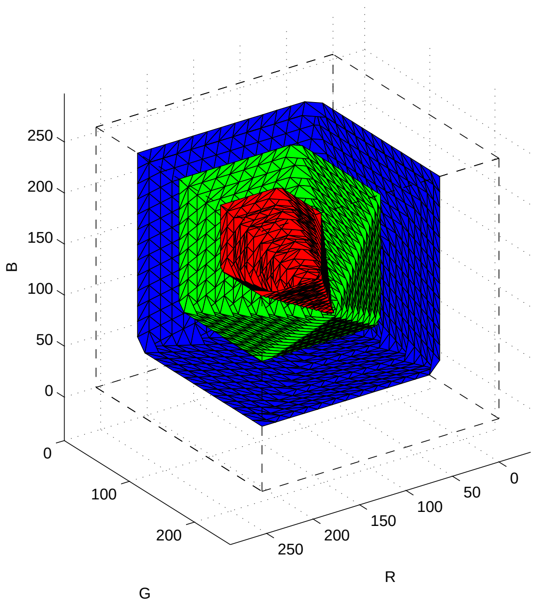
3.6. Comparison of detection methods

In this section, we compare speed and precision of the suggested detection methods. We use manually annotated video-sequences for evaluation. The training set contains 192 frames with both red and blue ball, while the evaluation sets consist of 100 frames with red ball and 100 frames with blue ball.

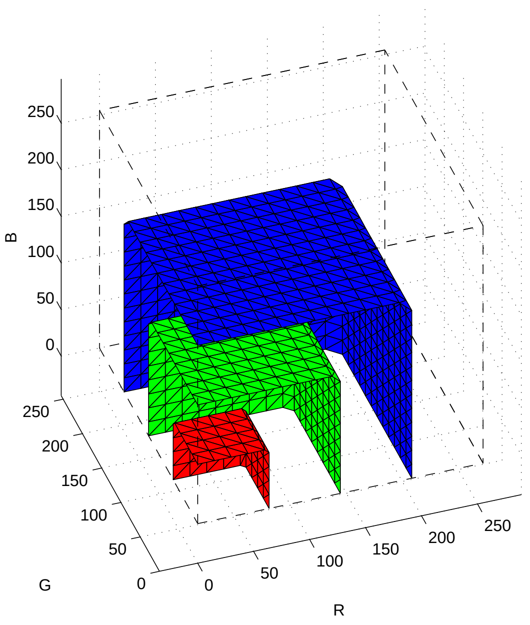
When evaluating the speed of the algorithms, we measure the time of processing of the 200 frames from the evaluation sets using MATLAB profiler. The measured time does not include initialization procedures of the detection methods like the boundary



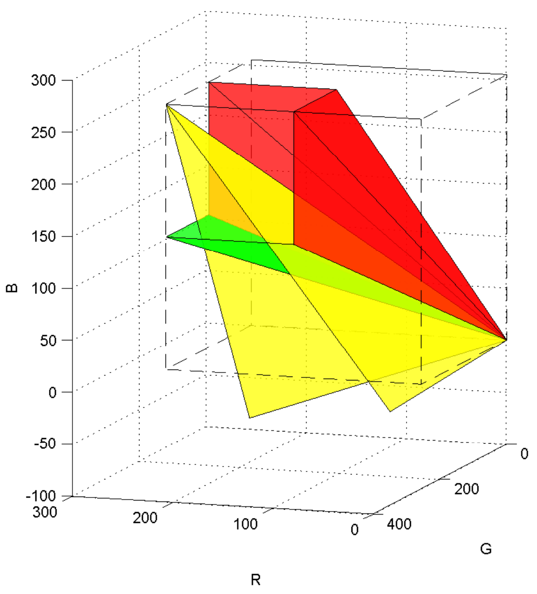
(a) *hue* isosurfaces in RGB space



(b) *saturation* isosurfaces in RGB space



(c) *value* isosurfaces in RGB space



(d) boundary plane selection

Figure 3.8. Hue and saturation isosurfaces in the RGB color space. Image (a) shows isosurfaces for $H = 0$ (yellow), $H = 0.25$ (red), $H = 0.5$ (green) and $H = 0.75$ (blue). Image (b) shows isosurfaces for $S = 0.25$ (red), $S = 0.5$ (green) and $S = 0.75$ (blue). Image (c) shows isosurfaces for $V = 0.25$ (red), $V = 0.5$ (green) and $V = 0.75$ (blue). Image (d) shows selection of applicable boundary planes for particular hue and saturation bounds $H_{\text{low}} = 0.1$, $H_{\text{upp}} = 0.3$ and $S_{\text{low}} = 0.5$ (the upper saturation bound was omitted). There are two yellow planes which stand for lower and upper hue limit, the green planes are a subset of saturation boundaries selected for thresholding and the red planes are a subset of saturation boundaries that were rejected because they are not necessary for the given range of hue.

Detection method	200 frames [s]	1 frame [ms]	Frame rate [Hz]
Background subtraction	0.99	4.9	202
Meanshift in RGB space	3.58	17.9	55
Meanshift in HSV space	8.08	40.4	24
Linear SVM	0.44	2.2	459
HSV thresholding	3.02	15.1	66
HSV thresholding (planes)	0.84	4.2	236

Table 3.1. Ball detection speed with ROI size 300 × 300 pixels and one ball. The computation times were measured in a 64-bit operating system running Intel Core i5 3550 @ 3.30 GHz.

	Median error [px]		Worst error [px]	
Detection method	red ball	blue ball	red ball	blue ball
Background subtraction	1.98	3.77	14.37	34.94
Meanshift in RGB space	3.01	11.85	6.65	27.57
Meanshift in HSV space	3.17	8.90	6.30	13.74
Linear SVM	1.16	3.19	3.05	11.11
HSV thresholding	1.59	1.66	3.13	4.74

Table 3.2. Error of ball detection measured in Euclidean distance of the detected position from the manually annotated ball centers.

plane computation for HSV feature thresholding, because these operations are performed just once and can be done in advance, so they are not time-critical. All of the methods use restricted ROI of size 300 × 300 pixels for detection. The results of speed evaluation for detection of one ball are summarized in Table 3.1. The necessary computational time for simultaneous detection of several balls approximately multiplies with the number of detected balls.

When we look at Table 3.1, we can see that the fastest method is the linear SVM, followed by the HSV thresholding in RGB space and background subtraction while the methods based on meanshift are too slow to be useful for our application. Although the computation should run faster without the profiler, the difference is at most in the order of percent. Note that the frame rates in Table 3.1 are only theoretical, because we only measured the ball detection computation time. In a practical application, we will have to grab the camera image and compute control actions besides the ball detection, so the resulting achievable frame rate will be lower.

The precision of detection is evaluated on the same video frames as the speed, for red and blue ball separately to show certain dependence of precision on ball color and camera settings. For good detection quality, it is necessary that the ball contrasts with the local part of the background, which is not always true. Especially the blue ball, which is relatively darker than the red one, blends into the dark background when it gets away from the area above the coil array. The precision of detection methods is compared in Table 3.2. The best results in terms of median and maximal error are achieved with HSV thresholding, which slightly outperforms the second-best linear SVM method in blue ball detection. We can notice that the detection of the red ball is apparently more precise with all algorithms because the red ball contrasts with the background better than the blue one. We should also emphasize that we compute position error with respect to manually annotated positions, which are not perfectly precise themselves, so even a precise detection can result in a non-zero error in the evaluation.

3.7. Camera calibration

When we want to use the computer vision system for visual feedback, we need to calibrate the camera to be able to transform between the image coordinates and real-world coordinates. The real-world coordinates represent the x- and y-coordinates of the ball on the surface of the platform, normalized so that we have unit distance between the centers of every two neighbouring coils of the array.

Although it is quite common calibrate the camera using a calibration image, which is usually a special high-contrast pattern, we can use the magnetic platform itself instead. We attract the ball to all coils of the platform by turning them on one-by-one. When the ball settles above a coil, we detect its position in several consecutive video frames and move on to another coil. This procedure gives us sets of corresponding image and scene points, which allow us to determine the parameters of a homography ([14]) between the image plane coordinates (x', y') and the normalized coordinates on the surface plane of the platform $(\hat{x}, \hat{y})$. The homography is a projective transformation

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}, \qquad \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = \frac{1}{w} \begin{bmatrix} u \\ v \end{bmatrix} \tag{3.19}$$

To find the homography parameters $\boldsymbol{H}$, we build up an overdetermined system of homogeneous linear equations

$$\boldsymbol{A}\boldsymbol{h} = \boldsymbol{0}, \tag{3.20}$$

where the matrix $\boldsymbol{A}$ contains measured corresponding points ordered as

$$\boldsymbol{A} = \begin{bmatrix} x'_1 & y'_1 & 1 & 0 & 0 & 0 & -\hat{x}_1 x'_1 & -\hat{x}_1 y'_1 & -\hat{x}_1 \\ 0 & 0 & 0 & x'_1 & y'_1 & 1 & -\hat{y}_1 x'_1 & -\hat{y}_1 y'_1 & -\hat{y}_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_N & y'_N & 1 & 0 & 0 & 0 & -\hat{x}_N x'_N & -\hat{x}_N y'_N & -\hat{x}_N \\ 0 & 0 & 0 & x'_N & y'_N & 1 & -\hat{y}_N x'_N & -\hat{y}_N y'_N & -\hat{y}_N \end{bmatrix}$$

and vector $\boldsymbol{h}$ is formed by stacking columns of the homography matrix $\boldsymbol{H}$. We use singular value decomposition (SVD) to find the approximate solution of the system of equations 3.20, which minimizes the quadratic error

$$\boldsymbol{h}^* = \underset{||\boldsymbol{h}||_2=1}{\arg\min} \left\{ \boldsymbol{h}^T \boldsymbol{A}^T \boldsymbol{A} \boldsymbol{h} \right\}. \tag{3.21}$$

The constraint on the norm of $\boldsymbol{h}$ for minimization is necessary to prevent selecting a zero vector. Decomposing $\boldsymbol{A}$ with SVD as $\boldsymbol{A} = \boldsymbol{U\Sigma V}^T$ gives the sought approximate solution $\boldsymbol{h}^*$ as the column of $\boldsymbol{V}$ corresponding to the smallest singular value on the diagonal of $\boldsymbol{\Sigma}$. When we reconstruct the matrix $\boldsymbol{H}$ from the approximate solution vector, we are able to transform the image plane coordinates to the platform plane coordinates according to expression 3.19.

3.8. Simulink implementation of ball detection

All the ball detection methods described above were first developed and tested in MATLAB. Afterwards, we converted the linear SVM and the HSV thresholding in the RGB color space to Simulink blocks. The other detection methods were rejected because they were either too slow (meanshift, HSV thresholding with colorspace conversion) or too imprecise (background subtraction has relatively good median error, but the worst errors are extremely high).

During the experiments, we acquire the video through the block From Video Device from Image Acquisition Toolbox in Simulink's normal mode because of the missing hardware Bayer interpolation support in the external mode. This solution is not deployable on the Speedgoat platform, but the software Bayer interpolation in Simulink would not be feasible anyway because of computational speed reasons.

The Simulink blocks for computer vision are implemented as Level-2 MATLAB S-Functions, which allow storage of the internal state of the detection algorithm between consecutive simulation time-steps. The use of Level-2 MATLAB S-Functions is possible thanks to the normal mode of model execution. For the external mode, we would have to provide a handwritten TLC file for each S-Function to support code generation. The TLC file specifies the structure of a Simulink block and its data and controls the C code generation process. Unfortunately, the TLC file is too complex to be written manually for as complicated S-Functions as the ones for ball detection.

The detection methods were also implemented as MATLAB Function blocks for potential future use in the external mode. The MATLAB Function blocks allow direct embedding of MATLAB code into models that generate C code, but they cannot store their internal state between consecutive simulation time-steps. The internal state of the detection algorithms is therefore stored in an external memory and is passed to the MATLAB Function block as an input argument every simulation period.

The Simulink implementations were profiled in the normal execution mode. The Simulink profiler measured longer execution times than the MATLAB profiler. Namely, we measured 5.5 ms per frame for the linear SVM and 4.6 ms per frame for the HSV thresholding in the RGB space, both of them implemented as Level-2 MATLAB S-Functions. We do not know how to explain that the difference for linear SVM is more remarkable. Furthermore, Simulink profiler revealed that the MATLAB function implementations are not suitable for the normal mode in Simulink, because their computation times are approximately ten times longer than those achieved with the MATLAB S-Functions. We think that this is caused by the fact that unlike the rest of the Simulink model, the MATLAB Functions are compiled even in normal execution mode, which makes passing of the arguments to the MATLAB function lengthy. In addition, we found out that just the transfer of the data from the framegraber to Simulink takes approximately 14 ms, so we cannot reach a higher frame rate than 50 Hz for a single ball or 40 Hz for two balls with the presented computation times.

3.9. Conclusions

We have chosen several suitable methods for ball detection in the video frames. All of the methods except for the background subtraction support simultaneous detection of more than one ball. The methods were implemented both in MATLAB and Simulink in a way that allows deployment on an external target, although the use with an external target is not possible so far due to the lack of hardware Bayer interpolation support. The Simulink blocks for ball detection were integrated in a Simulink library accessible from Simulink Library Browser, which ensures automatic update of the blocks in all referenced Simulink models after any change in the library. The library is available on the attached CD.

The detection methods were evaluated from the point of view of speed and precision. We have annotated sets of video frames for the purposes of ball detector training and evaluation.The speed was optimized with use of MATLAB profiler, which helped to identify bottlenecks of the algorithms and suggest improvements as in the case of HSV thresholding in the RGB color space.

An experiment with a ball in steady position has shown that the visual position measurement is less noisy than the touch foil measurement. While the standard deviation of the foil measurement is $\sigma = 0.375$ mm, the visually detected ball position does not have an approximately normal distribution, so it does not make sense to characterize it with standard deviation. The detected ball position is quantized due to the character of the detection methods, and the visually detected coordinates were either single values during the whole experiment or they oscillated between discrete values whose distance from each other was at most 0.2 mm.

The Simulink profiler discovered a significant delay of the visual position measurement. The delay consists of the camera's exposure time (8 ms), transfer of the data to Simulink (14 ms) and the detection computation time (approximately 5 ms), resulting in a total delay around 27 ms, which affects the performance of any feedback control.

Overall, the visual position measurement can be used for feedback control, but it is limited to the use with at most two balls, because the lengthy data transfer from the framegrabber to Simulink does not leave enough computation time for detection of more balls.

4. Reinforcement learning

This chapter is dedicated to the area of reinforcement learning which is a promising and still developing field of modern learning control techniques. We will describe the basic principles of reinforcement learning and take a look at the particular methods that were applied to the control of the magnetic platform.

The term reinforcement learning refers to a set of machine learning methods that combine experience-based learning process, which is typical for animals, with optimal and adaptive control. A fundamental publication in the area of reinforcement learning is [39], which introduces and explains the basic concepts of the broad field of reinforcement learning. There are several near-synonym terms to reinforcement learning in the literature, including approximate/asymptotic/adaptive dynamic programming (ADP, [34]), heuristic dynamic programming (HDP, [2]) or neuro-dynamic programming (NDP, [5]), altogether called Adaptive Critic Designs (ACD, [45]). All of the terms denote methods that are attempting to determine an optimal control policy in terms of long-term outcome by finding approximate solution to the Hamilton-Jacobi-Bellman equation. The aim is to find a mapping from particular situations to suitable actions, which allow achieving either a single-time goal (episodic task, e.g. a mobile robot reaching a desired location with minimal energy consumption), or maximizing the outcome of some continuing task according to a specific criterion (e.g. trajectory tracking with minimal error).

4.1. Basic terms

The reinforcement learning has a specific terminology. The learner is called an *agent*, which is continually interacting with an *environment* through some actions. An agent can be therefore understood as a controller and the environment as a controlled system. At every time step, the agent selects a suitable action u based on its previous experience and current situation, i.e. the observable state of the environment $\boldsymbol{x}$. The set of rules for action selection (mapping from environment's state to actions) is called *policy* and is usually denoted $h(\boldsymbol{x})$. By taking a particular action, the agent gets into a different situation and beside the information about the new state of the environment it also receives a *reward*, denoted $r(\boldsymbol{x}, u)$. The reward is a scalar numerical signal which indicates whether the agent reached its goal or how close to the goal it is and tells the agent how well or badly it performs. The agent incorporates the reward signal into its experience in order to improve its policy, attempting to maximize the long-term cumulative reward. For the continuing infinite-horizon tasks, it is necessary to weigh the reward exponentially with a *discount factor* $\gamma \in \langle 0, 1)$ to make the

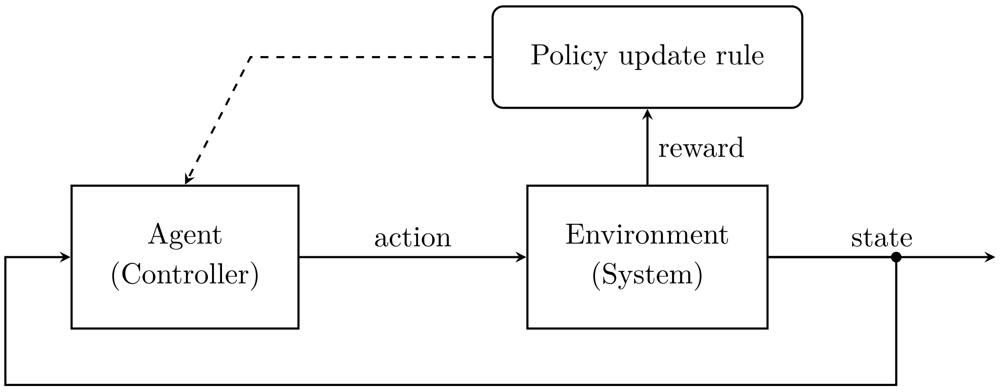


Figure 4.1. Reinforcement learning scheme. The agent interacts with the environment through taking some actions. The action outcomes are evaluated by a reward signal, which is used to update the agent's policy.

cumulative reward finite. The scheme of the agent–environment interaction is shown in Figure 4.1.

The particular reinforcement learning methods differ in the way how the agent projects the information from the reward signal into its policy. The experience of the agent is accumulated in a *value function*, which represents the expected cumulative reward received in the future when following a given policy. A *Q-function* or *state-action value function* $Q^h(\boldsymbol{x}, u)$ specifies the expected outcome of a trajectory starting from state $\boldsymbol{x}$ by taking action u and following the policy h onwards. A *V-function* or *state value function* $V^h(\boldsymbol{x}) = Q^h(\boldsymbol{x}, h(\boldsymbol{x}))$ is sometimes used as an alternative to the Q-function, assuming that already the starting action of the evaluated trajectory is selected using the policy h. Note that every value function is always related to a particular policy. While following a policy, the agent has to balance between exploitation of the policy (taking the best actions according to its experience) and exploration (trying new actions to find out whether or not they are better than the ones that already have been tried).

4.2. Markov Decision Process

The reinforcement learning requires some properties of the state of the environment observable by the agent to work properly ([39], Chapter 3.5). Although the observable state typically does not contain complete information about the environment, it should contain all the information from the environment's history, which is relevant for future transients. A state that retains all the relevant information is called Markovian. In reinforcement learning, we endeavor to get the environment's state which is at least approximately Markovian. As the decisions of the agent are based on the current state, it should carry enough information about the environment to allow making efficient decisions. Since we are interested in application of reinforcement learning to control of dynamic systems, it is a good idea to include the state of the dynamic system into the state of the environment available to the agent, because the state of the dynamic system has the Markov property.

When the observable state of the environment has the Markov property, reinforcement learning can be considered a Markov Decision Process (MDP), which provides a theoretical framework for the description of the algorithms. A formal definition found in [31] says that MDP (X, U, P, R) consists of a set of states X, a set of actions U, transition probabilities P and a cost function R. The transition probabilities $P : X \times U \times X \rightarrow [0; 1]$ are the conditional probabilities of transition to a state $\boldsymbol{x}' \in X$ given the current state $\boldsymbol{x}$ and taking action u. The cost function $R : X \times U \times X \rightarrow \mathbb{R}$ is the expected cost of transitioning from $\boldsymbol{x}$ to $\boldsymbol{x}'$ by taking u. In reinforcement learning terms, the reward signal can be taken as the negative value of some cost function. The main goal of the MDP is to find an optimal policy that minimizes the expected future cost, i.e. maximizes the future cumulative discounted reward

$$\mathcal{R}_t = \sum_{k=0}^{k_{term}} \gamma^k r(\boldsymbol{x}_{t+k+1}, u_{t+k+1}), \tag{4.1}$$

where the final time step k_{term} specifying the horizon of interest can be either finite (for episodic tasks) or infinite (for continuing tasks). The episodic tasks can be generalized to continuing tasks by defining an absorbing final state, which always transitions to itself with a zero reward. It is obviously necessary to use discount factor $\gamma < 1$ for infinite horizon tasks to ensure finite $\mathcal{R}_t$.

4.3. Taxonomy

In this section we would like to comment on the status of reinforcement learning amongst other machine learning methods as well as on the basic classes of reinforcement learning algorithms.

While it is usual to distinguish supervised and unsupervised machine learning methods, the reinforcement learning is standing between these two groups. In the supervised learning (also called "learning with a master"), we have to provide the learner with a learning set of input vectors labeled with desired outputs, so that it can precisely see its errors. Such labeling can make the learning process quite effective, but it also requires that the correct labeling is known in advance, so the designer – master must know the right solution. In the reinforcement learning, the learner – agent is responsible for the learning process itself, exploring the environment through executing some actions. The selection of the actions to perform is based on the information from the environment's state (an input vector) and agent's experience. However, the observed states of the environment are not labeled with any "correct" actions to take. The agent has to explore which actions are the correct ones in every particular situation and it can only observe the positive or negative effect of the taken actions through the environment transitions and a reward signal. The reward signal only indicates the goal being reached or some distance from the goal, which is much less informative than some labeling of input vectors and allows the designer to only know what to achieve, not how to achieve it. This is why reinforcement learning is sometimes quite descriptively referred to as “learning with a critic” ([20]) or “learning from interaction” ([39]).

If we divide the machine learning methods into model-based and model-free methods, we can find instances of reinforcement learning in both groups. A model can significantly accelerate the learning rate and improve the performance of the agent, especially with gradient actor-critic methods ([38]), because it can introduce some prior knowledge and structure to the learning process. There are also model-learning methods ([23]), which are sometimes confusingly called model-based in the literature, too. The model-free methods typically learn more slowly, but require less prior information about the controlled system, which is advantageous when a good model is not available.

The reinforcement learning methods themselves can be divided into several groups according to different criteria ([12]). All the methods can work either online or offline. The online methods adjust the parameters of the learner every few steps, while the offline methods wait with the update until the final goal is reached. Apparently, the online methods have a more complicated task than the offline ones, because they typically do not know the exact final outcome of the last decision at the time of parameter update, whereas the offline methods have a whole sequence of decisions and state transitions available for learning. The online methods are more suitable for control of dynamic systems, because it may take very long time for the offline methods before the agent reaches the goal during the initial trials. In addition, the ability to learn “on the go” makes the online methods applicable not only to episodic, but also to continuing tasks.

The reinforcement learning methods can be also split to on-policy and off-policy ones. The on-policy methods evaluate and improve directly the policy, which is used to make decisions, so they sacrifice optimal performance to exploration of the environment (e.g. the SARSA algorithm presented in [39] or actor-critic methods presented later in this section). In a stationary environment, the exploration can be attenuated over the time in, but in a time-varying environment the exploration level has to be maintained in order to adapt to possible changes. The off-policy methods like Q-learning ([39]) attempt to learn a deterministic optimal policy while following another one, which ensures sufficient amount of exploration.

4.4. Generalized Policy Iteration

The classical reinforcement learning methods estimate value functions in order to determine a policy. As the reinforcement learning originates from the area of discrete systems, the value functions and policies were originally often represented in some lookup tables, so the methods are sometimes called “tabular” in the literature.

The aim of reinforcement learning is to find an optimal policy in terms of long-term reward. An optimal policy $h^*(\boldsymbol{x})$ is a policy, whose expected long-term reward is greater than or equal to any other policy. Especially in discrete problems (discrete states, actions), there can exist more than one optimal policy. However, for all optimal policies, there exists a common optimal state value function $V^*(\boldsymbol{x})$ and an optimal state-action value function $Q^*(\boldsymbol{x}, u)$. The optimal value functions satisfy the Bellman optimality equation

$$V^*(\boldsymbol{x}) = \max_u \sum_{\boldsymbol{x}'} P^u_{\boldsymbol{x}\boldsymbol{x}'} \left[R^u_{\boldsymbol{x}\boldsymbol{x}'} + \gamma V^* \left(\boldsymbol{x}'\right)\right], \tag{4.2}$$

$$Q^*(\boldsymbol{x}, u) = \sum_{\boldsymbol{x}'} P^u_{\boldsymbol{x}\boldsymbol{x}'} \left[R^u_{\boldsymbol{x}\boldsymbol{x}'} + \gamma \max_{u'} Q^* \left(\boldsymbol{x}', u'\right)\right], \tag{4.3}$$

where $P^u_{\boldsymbol{x}\boldsymbol{x}'}$ is the probability of transitioning from state $\boldsymbol{x}$ to $\boldsymbol{x'}$ after taking action u, $R^u_{\boldsymbol{x}\boldsymbol{x}'}$ is the expected reward for the transition and γ is the discount factor. Note that there are two different conventions for optimization. In the computer science community, it is usual to consider rewards and maximize the performance index (cumulative return), while in the control community, it is common to consider costs and therefore minimize the performance index (cumulative cost). We will use the "reward–return" convention here to be consistent with the reinforcement learning terminology.

With a known optimal value function, it is already easy to find an optimal policy, because every optimal policy is greedy with respect to the corresponding optimal value function. The crucial contribution of the optimal value function is that it allows maximizing the long-term cumulative reward by local maximization of the value function.

The optimal value function can be estimated iteratively by applying a concept which is called Generalized Policy Iteration (GPI, [39]). The method combines two antagonistic interacting processes. The first process, called policy evaluation (see Algorithm 4.1), estimates the value function of the current policy from the observed environment states and rewards, which typically makes the policy non-greedy with respect to the estimate of the value function. The value function in Algorithm 4.1 is represented in a tabular form $V(\boldsymbol{x})$, which allows both read and write operations. The other process, called policy improvement, makes the policy greedy with respect to the latest estimate of the value function, which typically invalidates the current estimate of the value function. The combination of some form of policy evaluation with alternating or simultaneous policy improvement illustrated in Figure 4.2 is a fundamental concept of reinforcement learning.

Algorithm 4.1 Iterative policy evaluation

```
repeat
    $\Delta \leftarrow 0$
    for all $\boldsymbol{x} \in X$ do
        $v \leftarrow V(\boldsymbol{x})$
        $V(\boldsymbol{x}) \leftarrow \sum_{\boldsymbol{x}'} P^{h(\boldsymbol{x})}_{\boldsymbol{x}\boldsymbol{x}'} \left[ R^{h(\boldsymbol{x})}_{\boldsymbol{x}\boldsymbol{x}'} + \gamma V(\boldsymbol{x}')\right]$
        $\Delta \leftarrow \max\{\Delta, |v - V(\boldsymbol{x})|\}$
    end for
until $\Delta > \varepsilon$,    $\varepsilon$ is a small positive number
```

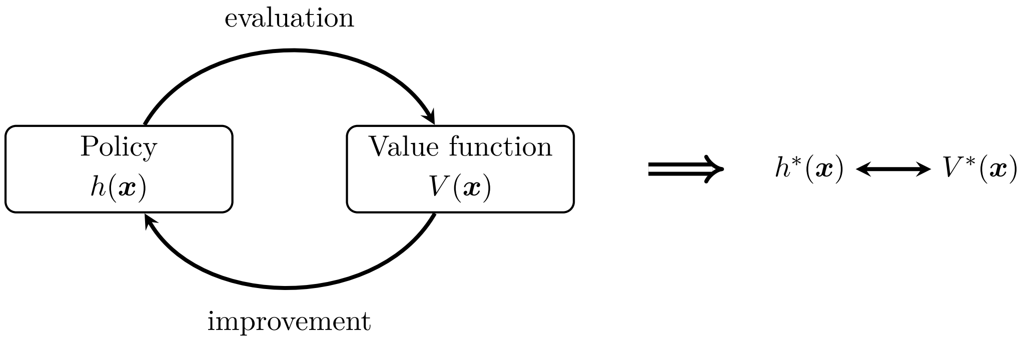


Figure 4.2. Generalized Policy Iteration alternates between making the value function consistent with the policy (policy evaluation) and making the policy greedy with respect to current value function (policy improvement). The optimal policy and value function are consistent with each other.

The two processes can interact at different levels of granularity. In the classical policy iteration, the policy improvement does not start until the policy evaluation converges to a solution. However, it is also possible to truncate the policy evaluation, which is itself an iterative procedure (see Algorithm 4.1), after a single update of the value function for each state, resulting in the so-called value iteration method. In an extreme case, which is referred to as asynchronous dynamic programming, the policy evaluation updates the value function entries only for one or several states before switching back to policy improvement. All of the above approaches eventually lead to convergence to the optimal value function and optimal policy, where the policy evaluation and improvement do not make any changes anymore, because an optimal policy is already greedy with respect to its value function.

4.5. Temporal difference

The classical exact GPI methods mentioned in the previous section require a precise model of the environment in a form of state transition probabilities, which can be difficult or even impossible to obtain. Although there are extensions available called Monte Carlo methods ([39]) that avoid the need for a model, they update value function estimates and policy on an episode-by-episode basis, which can become a problem when the episodes take a long time. The temporal difference (TD) methods overcome both need for a model and episode-by-episode learning problems. They allow a fully incremental step-by-step learning without a model, which is particularly suitable for online applications. The key idea of the temporal difference methods is to compare the actual received reward with its estimated prediction after every step k. The agent at state $\boldsymbol{x}_k$ applies an action u_k, receives reward r_{k+1}, observes the next state $\boldsymbol{x}_{k+1}$ and computes the temporal difference error

$$\delta_k = [r_{k+1}(\boldsymbol{x}_k, u_k) + \gamma V(\boldsymbol{x}_{k+1})] - V(\boldsymbol{x}_k). \tag{4.4}$$

If the agent was using a Q-function instead of V-function, it would be also necessary to choose the next action u_{k+1} for the temporal difference error computation as in the SARSA algorithm ([39])

$$\delta_k = [r_{k+1}(\boldsymbol{x}_k, u_k) + \gamma Q_k(\boldsymbol{x}_{k+1}, u_{k+1})] - Q_k(\boldsymbol{x}_k, u_k). \tag{4.5}$$

The temporal difference error is used to update the value function estimate. The general idea is to move the value function estimate against the temporal difference error

$$V_{k+1}(\boldsymbol{x}_k) = V_k(\boldsymbol{x}_k) + \alpha_k \delta_k, \tag{4.6}$$

where $\alpha_k \in (0; 1\rangle$ is the learning rate. The policy is usually updated in longer intervals and/or with lower learning rate than the value function, which is similar to the natural learning process – if some situation stops fitting our previous experience, we also go through it several more times before we ultimately adjust our behavior (policy) to that change.

It is noticeable that an agent in both cases 4.4 and 4.5 uses the estimate of value function for computation of temporal difference error. Therefore it updates a value estimate for some state on the basis of value estimate for some other state, which is called bootstrapping. Even though the updates are performed using imprecise data, it can be proven ([39]) that the learning process still converges to the optimal value function in the mean with a sufficiently small constant learning rate α_k and it is guaranteed to converge when the learning rate decreases in time and satisfies the stochastic approximation conditions

$$\sum_{k=1}^{\infty} \alpha_k = \infty, \qquad \sum_{k=1}^{\infty} \alpha_k^2 < \infty. \tag{4.7}$$

There is a specific group of temporal difference methods called *actor–critic* methods, which use two separate independent structures to represent the policy and value function. The structures are called an *actor* and a *critic*. The actor implements the policy, i.e. it selects the actions to be executed, while the critic evaluates the actions taken by the actor, which estimates a state value function. The main purpose of the critic is to produce a scalar temporal difference error signal, which is used to tune both of the structures simultaneously. The explicit representation of policy minimizes computational costs for large action spaces and is easily extensible from discrete to continuous environments.

Overall, the temporal difference methods are state of the art methods for online applications that do not need an environment model and have the ability to learn incrementally along trajectories. All of the methods we applied to the magnetic platform are based on the temporal difference.

4.6. Exploration

As already mentioned, the agent has to balance between the exploration (i.e. trying new actions and discovering new states) and the exploitation (using its previous experience to maximize the obtained reward). It is a difficult trade-off because the agent risks visiting undesirable states during the exploration beside discovering some policy improvements, which often decreases the reward. On the other hand, when the agent only exploits its experience in order to get maximal possible reward based on its current knowledge, it will stay stuck with a suboptimal policy. However, too much exploration makes it impossible to evaluate the current policy, because the effect of exploration can prevail over the effect of the actions intended by the current policy. In addition, the situation can be complicated by a stochastic environment, which requires trying an action in some state several times before getting a reliable estimate of the resulting reward.

A very common solution of the above dilemma is the ϵ-greedy action selection ([39]). In every step, the algorithm decides between a greedy action, which is selected with probability $(1-\epsilon)$ and exploratory (random) action, which is selected with probability ϵ. A greedy action is the one which has the highest expected reward according to the current estimate of the value function, exploratory action is any other one. When the action space is discrete, all of the actions have equal probabilities of being selected as an exploratory action. In the continuous action case, the exploratory action is selected as a random number drawn from a uniform distribution over the available interval of actions.

The ϵ-greedy action selection is not suitable for some tasks, because a bad exploratory action can completely destroy the task (e.g. the cliff walking example in [39]). In such cases a softmax action selection is more convenient. The softmax selection works similarly to the ϵ-greedy selection, but it uses a different than uniform distribution during exploratory action selection to discriminate the bad-rated actions in favor of the better-rated ones. [39] mentions the Boltzmann distribution as the most common one for softmax action selection.

In continuous action spaces, it is also possible to use an additive exploratory term in every time step ([38]). The action is then computed as a sum of a component given by current policy and the exploratory term, which is a random number, typically from normal distribution with zero mean value. It is a common practice to decrease the variance of the exploratory term in time to get a better performance after the agent has already explored the relevant areas of the environment.

4.7. Function approximation

The experience and policy of an agent in simple discrete environments can be easily represented in a tabular form. However, if the agent has to learn in a high-dimensional (or continuous) environment with large (or infinite) number of states, the tabular representation becomes inappropriate. Although the continuous environment could be discretized (sampled), a fine sampling would lead to a vast state space anyway. Several problems arise with the huge state spaces, jointly called the curse of dimensionality. First, the tabular representations would require huge amounts of memory. Second, the computational requirements for maintenance and searching through the tables would be too high and third, the probability of experiencing an exact state represented by every single table entry would drop close to zero, which is probably the most serious problem out of the three ones. Acting in vast and/or continuous environment, the agent needs the ability to generalize from experienced states to similar ones. The generalization ability is brought by function approximation.

Function approximation attempts to estimate a function in some region from observed samples, i.e. to generalize. The principle will be demonstrated on the approximation of a general function $f(\boldsymbol{x})$, which can be in practice a V-function, a Q-function or a policy function $h(\boldsymbol{x})$ for a continuous action space. An approximator $\hat{f}(\boldsymbol{x}, \boldsymbol{\theta})$ for the function $f(\boldsymbol{x})$ is not only function of the state $\boldsymbol{x}$, but also of a parameter vector $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_n]^T$. Training a function approximator is a supervised learning task which searches for the suitable parameters $\boldsymbol{\theta}$ that minimize the approximation error. The parametrization of the function approximator brings the generalization ability, because an adjustment of a parameter causes change of $\hat{f}(\boldsymbol{x}, \boldsymbol{\theta})$ in a whole region of the state-space.

There is a variety of function approximators used for reinforcement learning including the tile coding, fuzzy approximation, neural networks, and basis function approximation ([39], [37]). The basis function approximators (BFA) are probably the most commonly used option for control applications ([38], [43], [40], [11]), although some of the authors call them neural networks. Formally, the BFA can be described as a single-layer feedforward neural network with a single output neuron, but the principle is still the same.

4.7.1. Linear Basis Function Approximation

The linear BFA are parametric approximation methods, which are linear in their parameters. Each parameter θ_i has a corresponding basis function $\phi_i(\boldsymbol{x})$, which specifies the region of influence of the parameter in the state-space. Some parameters can therefore have a global influence, while the others are only local. The function approximation is computed as

$$\hat{f}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \phi_i(\boldsymbol{x})\theta_i = \boldsymbol{\phi}^T(\boldsymbol{x})\boldsymbol{\theta}, \tag{4.8}$$

where $\boldsymbol{\phi}(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \ldots, \phi_n(\boldsymbol{x})]^T$ is a vector of basis functions. The linearity in parameters θ_i is a big advantage that can be exploited for tuning of the parameter vector. With a known desired approximated value $f_k(\boldsymbol{x}_k)$, actual parameter vector estimate $\boldsymbol{\theta}_k$ and learning rate α, we get the new parameter vector estimate using a gradient-based update as

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \left[f_k(\boldsymbol{x}_k) - \hat{f}(\boldsymbol{x}_k, \boldsymbol{\theta}_k) \right] \nabla_{\boldsymbol{\theta}}\, \hat{f}(\boldsymbol{x}, \boldsymbol{\theta}) \Big|_{\boldsymbol{x}=\boldsymbol{x}_k, \boldsymbol{\theta}=\boldsymbol{\theta}_k}, \tag{4.9}$$

where the gradient of the approximated value function is directly the basis function vector thanks to the linearity in parameters

$$\nabla_{\boldsymbol{\theta}} \hat{f}(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\phi}(\boldsymbol{x}).$$

Beside the gradient-based parameter tuning, which is not limited only to linear function approximators, the linear least squares can be applied ([44]), either in batch or recursive form, solving a system of linear equations in form

$$\boldsymbol{\phi}^T(\boldsymbol{x}_k)\boldsymbol{\theta} = f_k(\boldsymbol{x}_k), \quad k = 1 \dots N, \tag{4.10}$$

where N is the length of a batch or $N \to \infty$ for recursive least squares.

The function approximation is a powerful tool that allows extension of reinforcement learning to continuous environments, although it is typically impossible to reach zero approximation error, which is a payoff for the generalization ability. The combination of nonzero approximation error with temporal difference learning can in extreme cases lead to divergence of the learning process ([18]) or to converge to a region instead of a point ([22]) including oscillations around the solution. However, it is usually possible to tune the parameters of the learning algorithm to reach convergence.

4.8. Online IRL control

The first of the applied methods we will describe here is the integral reinforcement learning (IRL, [31], [44], [43]). The method allows an online solution of an optimal control problem with only partial knowledge of the controlled system. The IRL is developed for a nonlinear dynamic system representable in form

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\boldsymbol{u}, \tag{4.11}$$

where $\boldsymbol{x}$ is the state of the controlled system and $\boldsymbol{u}$ is the control input. The optimal feedback control policy $\boldsymbol{u} = h(\boldsymbol{x})$ is sought using temporal difference learning on a state value function

$$V^h\left(\boldsymbol{x}\left(t\right)\right) = \int_t^\infty r\left(\boldsymbol{x}\left(\tau\right), \boldsymbol{u}(\tau)\right) \mathrm{d}\tau, \tag{4.12}$$

where the reward is defined as $r(\boldsymbol{x}, \boldsymbol{u}) = Q(\boldsymbol{x}) + \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u}$ with negative-definite function $Q(\boldsymbol{x})$ and negative-definite matrix $\boldsymbol{R}$. The optimal policy satisfies the continuous-time version of Bellman equation

$$0 = r\left(\boldsymbol{x}, h(\boldsymbol{x})\right) + \dot{V}^h(\boldsymbol{x}), \qquad \dot{V}^h(\boldsymbol{x}) = \left(\nabla V^h_{\boldsymbol{x}}\right)^T \left[f(\boldsymbol{x}) + g(\boldsymbol{x})h(\boldsymbol{x})\right] \tag{4.13}$$

For the case of a linear system we attempt to solve an LQR control problem, so the model and value function turn into

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \tag{4.14}$$

and

$$V^h\left(\boldsymbol{x}\left(t\right)\right) = \frac{1}{2}\int_t^\infty \left(\boldsymbol{x}^T(\tau)\boldsymbol{Q}\boldsymbol{x}(\tau) + \boldsymbol{u}^T(\tau)\boldsymbol{R}\boldsymbol{u}(\tau)\right) \mathrm{d}\tau \tag{4.15}$$

with both $\boldsymbol{Q}$ and $\boldsymbol{R}$ being negative-definite matrices. The usual linear quadratic performance index for LQ-optimal control assumes positive (semi–)definite weighting matrices, but we are using the reward convention instead of cost here, so the sign of the weights is opposite.

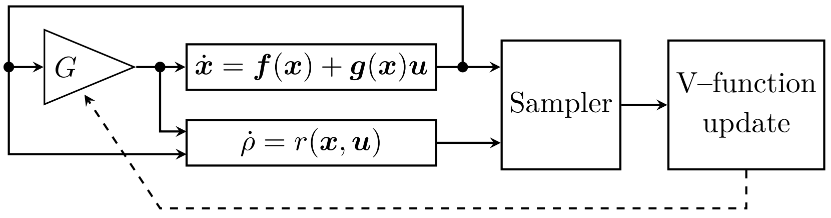

G
$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\boldsymbol{u}$
$\dot{\rho} = r(\boldsymbol{x}, \boldsymbol{u})$
Sampler
V–function update

Figure 4.3. Hybrid architecture of the IRL. The control loop and integral reinforcement can be implemented in continuous-time (e.g. in an analogue way), while the value function and feedback gain are updated in discrete time-steps.

4.8.1. Integral reinforcement

As the adaptive algorithm execution runs on a digital computer, it is inevitable to use some form of sampling. Although it would be possible to discretize the Equation 4.13 with sampling period T and sampled reward r_s to get its discrete form

$$0 = \frac{r_s\left(\boldsymbol{x}_k, h(\boldsymbol{x}_k)\right)}{T} + \frac{V^h(\boldsymbol{x}_{k+1}) - V^h(\boldsymbol{x}_k)}{T}, \tag{4.16}$$

the continuous Bellman equation would be only approximated in such case. The IRL can provide an exact variant called integral reinforcement form, which allows processing of sampled signals but preserves the continuous time. The Equation 4.13 is rewritten as

$$V^h\left(\boldsymbol{x}(t)\right) = \rho(t) + V^h\left(\boldsymbol{x}(t+T)\right), \qquad \rho(t) = \int_t^{t+T} r\left(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau)\right) \mathrm{d}\tau, \tag{4.17}$$

where $\rho(t)$ is called integral reinforcement. The IRL is a hybrid framework, where the integration of reward signal is performed in the continuous time domain, while the value function is updated at discrete moments with period T using sampled information from the system. After every update of value function, the policy is set greedy with respect to it as

$$h_k(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{R}^{-1}\boldsymbol{g}^T(\boldsymbol{x})\nabla V^{h_k}. \tag{4.18}$$

The feedback control can be performed in continuous time, too, with a piecewise constant control policy.

4.8.2. Linear system case

The value function is approximated with BFA in form 4.8. In the case of a linear system, we know that the optimal value function for an infinite horizon LQR problem is a quadratic form

$$V^{h^*}(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x} \approx \boldsymbol{\phi}^T(\boldsymbol{x})\boldsymbol{\theta}, \qquad \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{\theta} \in \mathbb{R}^{[n(n+1)/2]} \tag{4.19}$$

and the corresponding optimal control policy is a linear state feedback

$$h^*(\boldsymbol{x}) = -\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P}\boldsymbol{x}, \tag{4.20}$$

Algorithm 4.2 IRL policy iteration with value function aproximation

```
initialize an admissible policy $h(\boldsymbol{x})$ with a zero parameter vector $\boldsymbol{\theta}$
repeat
    collect a set of samples $\{[\boldsymbol{x}_i(t), \boldsymbol{x}_i(t+T), \rho_i(t)]\}$ using policy $h_k(\boldsymbol{x})$
    solve system of equations $\left[\boldsymbol{\phi}^T(\boldsymbol{x}_i(t)) - \boldsymbol{\phi}^T(\boldsymbol{x}_i(t+T))\right]\boldsymbol{\theta}_{k+1} = \rho_i(t)$
    reconstruct $\boldsymbol{P}_{k+1}$ from $\boldsymbol{\theta}_{k+1}$
    update the policy as $h_{k+1}(\boldsymbol{x}) = -\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P}_{k+1}\boldsymbol{x}$
until $||\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k+1}|| > \varepsilon$,    $\varepsilon$ is a small positive number
```

where $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ is a symmetric negative-semidefinite solution[1] of continuous-time algebraic Riccati equation (CARE)

$$\boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A} + \boldsymbol{Q} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} = \boldsymbol{0}. \tag{4.21}$$

Because the optimal value function is a quadratic function of state, it is advantageous to use quadratic functions of the state variables as the basis functions for BFA

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} x_1^2 & x_1x_2 & \dots & x_1x_n & \dots & x_2^2 & x_2x_3 & \dots & x_2x_n & \dots & \dots & x_n^2 \end{bmatrix}^T.$$

With the quadratic basis functions, the parameter vector $\boldsymbol{\theta}$ of the function approximator contains the estimates of the elements of the CARE solution $\boldsymbol{P}$ with the off-diagonal elements multiplied by a factor of 2, because we have

$$\boldsymbol{\phi}^T(\boldsymbol{x})\boldsymbol{\theta} \approx \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x} = \sum_{i=1}^{n} P_{ii}x_i^2 + 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} P_{ij}x_ix_j. \tag{4.22}$$

The learning process can be carried out either as policy iteration (Algorithm 4.2) or as value iteration (Algorithm 4.3). Both of the algorithms collect state, action and reward samples along the trajectory and use them to build up an overdetermined system of equations

$$\boldsymbol{\Gamma}\boldsymbol{\theta} = \boldsymbol{z} \tag{4.23}$$

where the index i in both algorithms can be understood as an equation index, because the i-th equation of the system of equations yields from the i-th collected set of samples $\{[\boldsymbol{x}_i(t), \boldsymbol{x}_i(t+T), \rho_i(t)]\}$.

The policy iteration has more strict requirements on the initial policy than the value iteration. The initial policy for policy iteration must be admissible, which means that it is stabilizing and yields a finite discounted reward. The value iteration does not necessarily need an admissible initial policy to converge, but for a real application it is reasonable that it is admissible, too, because a destabilization may e.g. damage the controlled system. Another difference between the two variants is that the policy iteration has a quadratic rate of convergence to the optimal solution, while the convergence rate of the value iteration is not defined and typically slower.

The IRL is able to solve the CARE online without the knowledge of the full system dynamics. We can see that both algorithms 4.2 and 4.3 exploit only knowledge of the

[1]When $\boldsymbol{Q}$ is negative-definite or $(\boldsymbol{A}, \boldsymbol{Q})$ is an observable pair, $\boldsymbol{P}$ is negative-definite.

Algorithm 4.3 IRL value iteration with value function aproximation

```
initialize an arbitrary policy $h(\boldsymbol{x})$ with a consistent parameter vector $\boldsymbol{\theta}$
repeat
    collect a set of samples $\{[\boldsymbol{x}_i(t), \boldsymbol{x}_i(t+T), \rho_i(t)]\}$ using policy $h_k(\boldsymbol{x})$
    solve system of equations  $\boldsymbol{\phi}^T(\boldsymbol{x}_i(t))\boldsymbol{\theta}_{k+1} = \rho_i(t) + \boldsymbol{\phi}^T(\boldsymbol{x}_i(t+T))\boldsymbol{\theta}_k$
    reconstruct $\boldsymbol{P}_{k+1}$ from $\boldsymbol{\theta}_{k+1}$
    update the policy as $h_{k+1}(\boldsymbol{x}) = -\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P}_{k+1}\boldsymbol{x}$
until $||\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k+1}|| > \varepsilon$,   $\varepsilon$ is a small positive number
```

input matrix $\boldsymbol{B}$ of the controlled system. In addition, the IRL can also adapt to the changes of the system's dynamics during the learning process. A disadvantage of the proposed method is that it requires a full state information to learn and control the system, which can become a problem when the full state cannot be measured directly and the model of the system is unknown. For mechanical systems like our ball on the platform, we are often able to measure the position, but the information about the velocity is missing, so we have to apply a filtering or numerical differentiation to compute the velocity from the position data. Output feedback variants of policy and value iteration have been already developed in [30], which do not need the full state information. However, they assume a deterministic (i.e. noise-free) linear time-invariant system, so they are not applicable for a real model.

Another drawback of the IRL is the need for persistent excitation of the controlled system to prevent the system of equations 4.23 from becoming rank-deficient, which would lead to ambiguous parameter vector $\boldsymbol{\theta}$. The issue is addressed more in detail in Section 6.1.

4.9. Online LSPI bang-off-bang control

The second method selected for the control of the platform is the online least-squares policy iteration (LSPI) developed in [11]. It is an online algorithm acting in a continuous state-space, discrete action space and discrete time with presented successful results in control of a real physical model of an underactuated inverted pendulum. Unlike the IRL presented in the previous section, the online LSPI does not presume any particular structure of the controlled system and considers a single-input controlled dynamic system in a general form

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, u_k), \qquad u_k = h(\boldsymbol{x}_k). \tag{4.24}$$

The policy iteration is run with a Q-function approximated with a BFA

$$Q(\boldsymbol{x}, u) \approx \hat{Q}(\boldsymbol{x}, u) = \boldsymbol{\phi}^T(\boldsymbol{x}, u)\boldsymbol{\theta}, \quad \boldsymbol{\theta} \in \mathbb{R}^n \tag{4.25}$$

which yields a subspace S, the subspace of representable Q-functions. Because we have a discrete action space, with three control actions available for the bang-off-bang control, the Q-function can be represented as a union of three separate V-functions, one for each control action. The BFA uses a joint vector of basis functions

$\bar{\boldsymbol{\phi}}(\boldsymbol{x}) = \left[\bar{\phi}_1(\boldsymbol{x}), \ldots, \bar{\phi}_{n/3}(\boldsymbol{x})\right]$ for all control actions, learning the coefficients θ_i for every action separately. This approach leads to a stacked basis function vector

$$\boldsymbol{\phi}(\boldsymbol{x}, u) = \left[\; \mathcal{I}(u = u_1) \cdot \bar{\boldsymbol{\phi}}^T(\boldsymbol{x}), \;\; \mathcal{I}(u = u_2) \cdot \bar{\boldsymbol{\phi}}^T(\boldsymbol{x}), \;\; \mathcal{I}(u = u_3) \cdot \bar{\boldsymbol{\phi}}^T(\boldsymbol{x}) \;\right]^T \tag{4.26}$$

with an indicator function $\mathcal{I}$, which is 1 when the equality in its argument holds and 0 otherwise. Note that the joint basis function vector is independent on the control action. The dependence of $\hat{Q}(\boldsymbol{x}, u)$ on the control action is introduced by the indicator function $\mathcal{I}$.

4.9.1. Incremental parameter vector estimation

The approximate Q-function is computed using a projection-based policy evaluation ([4]). The "projection" refers to the fact that the method solves the Bellman equation 4.3 projected into the subspace of representable Q-functions S. At every iteration k of the algorithm, the current approximation of the Q-function $\boldsymbol{\phi}^T\boldsymbol{\theta}_k$ is combined with reward r_k to produce a sample of the estimated Q-function. The samples collected in consecutive iterations usually characterize a function that does not lie in the subspace S, so they have to be projected into it using weighted least-squares projection $P^{\boldsymbol{w}}$ in order to compute a new approximation $\boldsymbol{\phi}^T\boldsymbol{\theta}_{k+1}$. The process leads to projected Bellman equation

$$\hat{Q}(\boldsymbol{x}, u) = P^{\boldsymbol{w}}\left(r(\boldsymbol{x}, u) + \gamma\hat{Q}(\boldsymbol{x}, u)\right), \tag{4.27}$$

which can be expressed in form of a weighted least-squares problem suitable for incremental updates

$$\boldsymbol{\Gamma}\boldsymbol{\theta} = \boldsymbol{z} + \gamma\boldsymbol{\Lambda}\boldsymbol{\theta}. \tag{4.28}$$

Parameters $\boldsymbol{\Gamma}$ and $\boldsymbol{\Lambda}$ are square matrices from $\mathbb{R}^{n\times n}$ and $\boldsymbol{z}$ is a vector from $\mathbb{R}^n$. The parameters are initialized with zeros except for $\boldsymbol{\Gamma}$, which is initialized as a small multiple of an identity matrix $\varepsilon\boldsymbol{I}^{n\times n}$, $\varepsilon > 0$ to guarantee its invertibility. The weighting of the least-squares is hidden in the way how $\boldsymbol{\Gamma}$, $\boldsymbol{\Lambda}$ and $\boldsymbol{z}$ are updated. Along the controlled system's trajectory, as the control policy $h(\boldsymbol{x})$ is being applied, a set of data $\{\boldsymbol{x}_k, u_k, \boldsymbol{x}_{k+1}, r(\boldsymbol{x}_k, u_k)\}$ is collected every time step k. The collected data set is used for an incremental update of the least-squares problem 4.28 as

$$\begin{aligned}
\boldsymbol{\Gamma}_{k+1} &= \boldsymbol{\Gamma}_k + \boldsymbol{w}_k\boldsymbol{\phi}^T(\boldsymbol{x}_k, u_k), \\
\boldsymbol{\Lambda}_{k+1} &= \boldsymbol{\Lambda}_k + \boldsymbol{w}_k\boldsymbol{\phi}^T(\boldsymbol{x}_{k+1}, h(\boldsymbol{x}_{k+1})), \\
\boldsymbol{z}_{k+1} &= \boldsymbol{z}_k + \boldsymbol{w}_k r(\boldsymbol{x}_k, u_k)
\end{aligned} \tag{4.29}$$

with weight vector $\boldsymbol{w}_k = \boldsymbol{\phi}(\boldsymbol{x}_k, u_k)$. Note that the IRL solution in Section 4.8.2 uses BFA as well, but the approximator can fit the sought quadratic form precisely, so it does not require any projection. It is also noticeable that in LSPI, the data points collected over the time are accumulated in the parameters $\boldsymbol{\Gamma}$, $\boldsymbol{\Lambda}$ and $\boldsymbol{z}$ using summation to combine the previously collected and latest data. This is a significant difference from the IRL approach in 4.23, where every data sample generates one equation and the previously collected data are forgotten after every policy update. Unlike the IRL, the online LSPI retains the past data, which does not allow adaptation to system parameter changes during learning, but does not require persistence of excitation between every two consecutive policy updates. Another positive fact is that the computational complexity of solving problem 4.28 depends only on the length of the BFA parameter vector $\boldsymbol{\theta}$, not on the amount of data observed.

4.9.2. Policy updates

The algorithm uses an ϵ-greedy control policy with respect to the latest estimate of the Q-function. The exploration rate ϵ decreases exponentially over time, which ensures sufficient amount of exploration at the beginning of the learning process and dominant exploitation of the control policy in the later learning phases when the policy is already close to an optimal one. It is necessary to choose such value of decay rate for ϵ that the exploration covers a time interval long enough for learning a good policy. The algorithm would probably stay stuck on a bad and/or suboptimal policy with a too fast decrease of ϵ. On the other hand, a too slow decrease would unnecessarily deteriorate the performance during learning.

Every K transitions, the system of equations 4.28 is solved to update the BFA parameter vector $\boldsymbol{\theta}$ and thus also the policy. The constant K is a tuning parameter of the algorithm and it can vary from 1 to several thousands, defining the period of policy updates. The extreme variant with $K = 1$ which updates the BFA and policy after every transition is called fully optimistic, because it believes that the most recent sample of data (starting and resulting state, action and reward) is representative enough for an update, which is not necessarily true in stochastic environments. When K is set in the order of thousands, the online LSPI gets closer to its offline variant, which fully evaluates the current policy before making an update.

The greedy policy after l-th update of the BFA can be expressed as

$$h_{l+1}(\boldsymbol{x}) = \arg\max_{u} \left\{ \hat{Q}_l(\boldsymbol{x}, u) \right\}. \tag{4.30}$$

Note that the policy is not represented explicitly, but it is computed on demand after measuring a particular state, because an explicit representation may be difficult in continuous state-space and the greedy action computation 4.30 is not computationally demanding.

The online LSPI seems to be a powerful tool for real-time online learning and control. Its capabilities were experimentally demonstrated in [11] on a real model of an inverted pendulum, which was swung up and stabilized in the upper unstable position. The advantage of LSPI is that it does not assume any specific structure or knowledge about the controlled system except for the at least approximately Markovian property of the system's state. In addition, the simplicity of the bang-off-bang control could be appealing for direct coil currents control in the future.

4.10. Online EBAC control

The third method for the Magman control we will present is the energy-balancing actor-critic control (EBAC, [38]), which combines passivity based control (PBC) for port-Hamiltonian systems with actor-critic reinforcement learning control. Although the method requires a model of the controlled system, which seems to oppose to our original aim of implementing a model-free learning controller it is appealing mainly from two reasons. First, when the previously described RL methods did not work very well, we were looking for a method that would effectively incorporate prior knowledge about the controlled system into the learning process. Second, although the EBAC uses a model knowledge, it is claimed to be robust to unmodeled nonlinearities like actuator saturation.

4.10.1. Port-Hamiltonian control paradigm

Port-Hamiltonian paradigm is a specific way of representing, analyzing and controlling dynamic systems. A detailed description can be found e.g. in [16]. The paradigm is based on the description of the total energy stored in the system and exchanged with its surrounding through ports. It is quite close to the bond graph paradigm. An input-state-output port-Hamiltonian system is represented as

$$\begin{aligned} \dot{\boldsymbol{x}} &= [\boldsymbol{J}(\boldsymbol{x}) - \boldsymbol{R}(\boldsymbol{x})]\,(\nabla_{\boldsymbol{x}} H(\boldsymbol{x})) + \boldsymbol{g}(\boldsymbol{x})\boldsymbol{u}, \\ \boldsymbol{y} &= \boldsymbol{g}^T(\boldsymbol{x})\,(\nabla_{\boldsymbol{x}} H(\boldsymbol{x}))\,. \end{aligned} \tag{4.31}$$

with a state vector $\boldsymbol{x} \in \mathbb{R}^n$, control input $\boldsymbol{u} \in \mathbb{R}^m$ and output $\boldsymbol{y} \in \mathbb{R}^m$. The interconnection matrix $\boldsymbol{J}(\boldsymbol{x})$ and damping matrix $\boldsymbol{R}(\boldsymbol{x})$ are mappings from the state space $\mathbb{R}^n$ to the space $\mathbb{R}^{n\times n}$. The interconnection matrix is antisymmetric, i.e. $\boldsymbol{J}(\boldsymbol{x}) = -\boldsymbol{J}(\boldsymbol{x})^T$, while the damping matrix is symmetric and positive semi-definite. The Hamiltonian $H(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ expresses the total energy stored in the system and $\boldsymbol{g}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ is a full-rank input matrix. The input and output are conjugated variables whose product is power. The change of the energy accumulated in the system is given by the difference of dissipated and added power, which is expressed by the power balance equation

$$\dot{H}(\boldsymbol{x}) = (\nabla_{\boldsymbol{x}} H(\boldsymbol{x}))^T \dot{\boldsymbol{x}} = -\left[(\nabla_{\boldsymbol{x}} H(\boldsymbol{x}))^T \boldsymbol{R}(\boldsymbol{x})\,(\nabla_{\boldsymbol{x}} H(\boldsymbol{x}))\right] + \boldsymbol{u}^T \boldsymbol{y} \tag{4.32}$$

where the term in brackets expresses the power dissipated in the open-loop and the term $\boldsymbol{u}^T\boldsymbol{y}$ the added power supplied through the input port.

The objective of energy-balancing passivity-based system control (EB-PBC) is to design such feedback for the controlled system that its new Hamiltonian function $H_\mathrm{d}(\boldsymbol{x})$ has the minimum located at a desired equilibrium $\boldsymbol{x}^*$, i.e. to make the system passive with respect to a desired Hamiltonian function, which describes the total energy of the closed-loop system. In addition, the feedback also has to ensure a desired damping $\boldsymbol{R}_\mathrm{d}(\boldsymbol{x})$. The task is accomplished by means of energy shaping and damping injection, leading to a closed-loop system

$$\begin{aligned} \dot{\boldsymbol{x}} &= [\boldsymbol{J}(\boldsymbol{x}) - \boldsymbol{R}_\mathrm{d}(\boldsymbol{x})] (\nabla_{\boldsymbol{x}} H_\mathrm{d}(\boldsymbol{x})), && (4.33)\\ \dot{H}_\mathrm{d}(\boldsymbol{x}) &= -(\nabla_{\boldsymbol{x}} H_\mathrm{d}(\boldsymbol{x}))^T \boldsymbol{R}_\mathrm{d}(\boldsymbol{x}) (\nabla_{\boldsymbol{x}} H_\mathrm{d}(\boldsymbol{x})). && (4.34) \end{aligned}$$

The control policy consists of an energy shaping part $h_\mathrm{e}(\boldsymbol{x})$ and a damping injection part $h_\mathrm{d}(\boldsymbol{x})$. The energy shaping part ensures that the minimum of the desired Hamiltonian function coincides with the desired equilibrium. The analytic expression for the energy shaping part presented in [38] is

$$\begin{aligned} h_\mathrm{e}(\boldsymbol{x}) &= \boldsymbol{g}^\dagger(\boldsymbol{x}) \boldsymbol{F}(\boldsymbol{x}) (\nabla_{\boldsymbol{x}} H_\mathrm{a}(\boldsymbol{x})), && (4.35)\\ \boldsymbol{g}^\dagger(\boldsymbol{x}) &= \left(\boldsymbol{g}^T(\boldsymbol{x})\boldsymbol{g}(\boldsymbol{x})\right)^{-1} \boldsymbol{g}^T(\boldsymbol{x}), \\ \boldsymbol{F}(\boldsymbol{x}) &= \boldsymbol{J}(\boldsymbol{x}) - \boldsymbol{R}(\boldsymbol{x}), \end{aligned}$$

where $H_\mathrm{a}(\boldsymbol{x}) = H_\mathrm{d}(\boldsymbol{x}) - H(\boldsymbol{x})$ is the added energy function, so its time derivative $\dot{H}_\mathrm{a}(\boldsymbol{x})$ is the power added by the controller through the system's input port. The added energy function solves the set of partial differential equations[2]

$$\begin{bmatrix} \boldsymbol{g}^\perp(\boldsymbol{x})\boldsymbol{F}^T(\boldsymbol{x}) \\ \boldsymbol{g}^T(\boldsymbol{x}) \end{bmatrix} (\nabla_{\boldsymbol{x}} H_\mathrm{a}(\boldsymbol{x})) = 0, \qquad \boldsymbol{g}^\perp(\boldsymbol{x})\boldsymbol{g}(\boldsymbol{x}) = 0. \tag{4.36}$$

The damping injection part of the control policy feeds back the output of the energy-shaped system to ensure the desired damping $\boldsymbol{R}_\mathrm{d}(\boldsymbol{x}) = \boldsymbol{R}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\boldsymbol{K}(\boldsymbol{x})\boldsymbol{g}^T(\boldsymbol{x})$ with $\boldsymbol{K}(\boldsymbol{x})$ being a symmetric matrix. Although the traditional EB-PBC assumes positive-semidefinite matrix $\boldsymbol{K}(\boldsymbol{x})$, it can be advantageous to relax this constraint to be able to supply energy into the controlled system through the damping term, as shown in the inverted pendulum swing-up [38]. When the matrix $\boldsymbol{K}(\boldsymbol{x})$ is known, the damping injection can be expressed as

$$h_\mathrm{d}(\boldsymbol{x}) = -\boldsymbol{K}(\boldsymbol{x}) \left[\boldsymbol{g}^T(\boldsymbol{x}) (\nabla_{\boldsymbol{x}} H_\mathrm{d}(\boldsymbol{x}))\right]. \tag{4.37}$$

The overall control policy for the energy-balancing passivity-based control is obtained as the sum of energy shaping and damping injection part

$$h(\boldsymbol{x}) = h_\mathrm{e}(\boldsymbol{x}) + h_\mathrm{d}(\boldsymbol{x}). \tag{4.38}$$

4.10.2. Actor-critic for energy balancing control

The contribution of the actor-critic reinforcement learning for energy balancing [38] is that it does not require explicit specification of the desired Hamiltonian function and damping matrix neither the analytic solution of partial differential equations 4.36 or finding the damping injection matrix. The control objective is specified by the reward signal instead and the reinforcement learning is used to find the corresponding parameters $H_\mathrm{d}(\boldsymbol{x})$ and $\boldsymbol{K}(\boldsymbol{x})$. The EBAC framework uses one critic and two independent actors that learn in trials. The critic approximates a V-function, one of the actors is

[2]The vector $\boldsymbol{g}^\perp(\boldsymbol{x})$ is the full-rank left annihilator of $\boldsymbol{g}(\boldsymbol{x})$.

responsible for the energy shaping part of the policy, learning the desired Hamiltonian function, and the other one serves for the damping injection, learning the damping matrix.

The critic is a BFA (4.8) in form $\hat{V}(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\phi}_\mathrm{c}^T(\boldsymbol{x})\boldsymbol{\theta}$. The critic parameter vector is tuned using temporal difference gradient descent method with eligibility traces [39]. The eligibility traces attempt to speed up the learning by retaining information about the previously visited states and propagating the temporal difference signal not only to the value of the last visited state but also to the preceding ones. Although there are empirical results which claim that eligibility traces can be counterproductive for some tasks ([41]), the authors of [38] find them helpful for the inverted pendulum and we have a positive experience for the ball, too. The incremental parameter vector update for the critic with temporal difference δ and eligibility traces $\boldsymbol{e}$ is

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_\mathrm{c}\delta_{k+1}\boldsymbol{e}_{k+1}, \tag{4.39}$$

where α_c is the critic learning rate and the temporal difference and eligibility traces are defined as

$$\begin{aligned} \delta_{k+1} &= r(\boldsymbol{x}_{k+1}, u_k) + \gamma\boldsymbol{\phi}_\mathrm{c}^T(\boldsymbol{x}_{k+1})\boldsymbol{\theta}_k - \boldsymbol{\phi}_\mathrm{c}^T(\boldsymbol{x}_k)\boldsymbol{\theta}_k, \\ \boldsymbol{e}_{k+1} &= \gamma\lambda\boldsymbol{e}_k + \boldsymbol{\phi}_\mathrm{c}(\boldsymbol{x}_k), \end{aligned} \tag{4.40}$$

with reward signal r, discount factor γ and eligibility trace decay rate $\lambda \in \langle 0; 1)$.

The energy shaping actor is constrained by conditions arising from the system of partial differential equations 4.36, which can be rewritten as

$$\underbrace{\begin{bmatrix} \boldsymbol{g}^\perp(\boldsymbol{x})\boldsymbol{F}^T(\boldsymbol{x}) \\ \boldsymbol{g}^T(\boldsymbol{x}) \end{bmatrix}}_{\boldsymbol{A}(\boldsymbol{x})} [(\nabla_{\boldsymbol{x}} H_\mathrm{d}(\boldsymbol{x})) - (\nabla_{\boldsymbol{x}} H(\boldsymbol{x}))] = 0. \tag{4.41}$$

The matrix $\boldsymbol{A}(\boldsymbol{x})$ typically has a kernel with basis $\boldsymbol{N}(\boldsymbol{x}) \in \mathbb{R}^{n\times b}$ so 4.41 can be expressed as

$$(\nabla_{\boldsymbol{x}} H_\mathrm{d}(\boldsymbol{x})) - (\nabla_{\boldsymbol{x}} H(\boldsymbol{x})) = \boldsymbol{N}(\boldsymbol{x})\boldsymbol{a}, \quad \boldsymbol{A}(\boldsymbol{x})\boldsymbol{N}(\boldsymbol{x}) = 0,\ \boldsymbol{a} \in \mathbb{R}^b. \tag{4.42}$$

The state vector $\boldsymbol{x}$ of the system can be reordered and split into two components as $\boldsymbol{x} = \begin{bmatrix} \boldsymbol{w}^T & \boldsymbol{z}^T \end{bmatrix}^T$, where the vector $\boldsymbol{w}$ corresponds with the non-zero rows of $\boldsymbol{N}(\boldsymbol{x})$ and the vector $\boldsymbol{z}$ with the zero ones. After the reordering, 4.42 turns into

$$\begin{bmatrix} \nabla_{\boldsymbol{w}} H_\mathrm{d}(\boldsymbol{x}) \\ \nabla_{\boldsymbol{z}} H_\mathrm{d}(\boldsymbol{x}) \end{bmatrix} - \begin{bmatrix} \nabla_{\boldsymbol{w}} H(\boldsymbol{x}) \\ \nabla_{\boldsymbol{z}} H(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{N}_{\boldsymbol{w}}(\boldsymbol{x}) \\ 0 \end{bmatrix} \boldsymbol{a}, \tag{4.43}$$

from where it is already obvious that the energy shaping can be done only on state variables $\boldsymbol{w}$, because 4.43 requires that $\nabla_{\boldsymbol{z}} H_\mathrm{d}(\boldsymbol{x}) = \nabla_{\boldsymbol{z}} H(\boldsymbol{x})$. The desired Hamiltonian function is therefore parametrized as a combination of the original Hamiltonian function and a BFA 4.8 acting on the energy-shaping state variables

$$\hat{H}_\mathrm{d}(\boldsymbol{x}, \boldsymbol{\xi}) = H(\boldsymbol{x}) + \boldsymbol{\phi}_\mathrm{e}^T(\boldsymbol{w})\boldsymbol{\xi}, \quad \boldsymbol{\phi}_\mathrm{e}(\boldsymbol{w}), \boldsymbol{\xi} \in \mathbb{R}^d, \tag{4.44}$$

which implicitly satisfies the conditions imposed by 4.41.

The damping injection is parametrized with a variant of BFA 4.8 acting on the whole state. However, the form 4.8 approximates only a scalar function, while $\boldsymbol{K}(\boldsymbol{x})$ is a matrix in general, so there is a separate approximator for each element of $\boldsymbol{K}(\boldsymbol{x})$

$$\hat{K}_{ij}(\boldsymbol{x}, \boldsymbol{\Psi}_{ij}) = \boldsymbol{\phi}_{\mathrm{d}}^T(\boldsymbol{x})\boldsymbol{\Psi}_{ij}, \quad \boldsymbol{\phi}_{\mathrm{d}}(\boldsymbol{x}), \boldsymbol{\Psi}_{ij} \in \mathbb{R}^e \tag{4.45}$$

The symmetry of matrix[3] $\hat{\boldsymbol{K}}(\boldsymbol{x}, \boldsymbol{\Psi})$ is enforced by setting $\boldsymbol{\Psi}_{ij} = \boldsymbol{\Psi}_{ji}$. The control policy 4.38 with the use of approximators 4.44 and 4.45 changes to

$$\begin{aligned} h(\boldsymbol{x}, \boldsymbol{\xi}, \boldsymbol{\Psi}) &= \boldsymbol{g}^\dagger(\boldsymbol{x})\boldsymbol{F}(\boldsymbol{x}) \begin{bmatrix} (\nabla_{\boldsymbol{w}}\boldsymbol{\phi}_{\mathrm{e}}(\boldsymbol{w}))^T \boldsymbol{\xi} \\ 0 \end{bmatrix} \\ &- \hat{\boldsymbol{K}}(\boldsymbol{x}, \boldsymbol{\Psi})\boldsymbol{g}^T(\boldsymbol{x}) \begin{bmatrix} (\nabla_{\boldsymbol{w}}\boldsymbol{\phi}_{\mathrm{e}}(\boldsymbol{w}))^T \boldsymbol{\xi} + \nabla_{\boldsymbol{w}} H(\boldsymbol{x}) \\ \nabla_{\boldsymbol{z}} H(\boldsymbol{x}) \end{bmatrix} \end{aligned} \tag{4.46}$$

The learning of actor and critic approximators requires sufficient exploration, which is ensured by extending the control action with a random additive component $\Delta\hat{u}$ drawn from a normal distribution $\mathcal{N}(0, \sigma^2)$. The variance of the distribution is chosen with respect to the available range of control action, usually around 50% of the maximal available amplitude. At every time step k, a control action is generated

$$\hat{u}_k = h(\boldsymbol{x}_k, \boldsymbol{\xi}_k, \boldsymbol{\Psi}_k) + \Delta\hat{u}_k. \tag{4.47}$$

Afterwards, the control action $\hat{u}_k$ is saturated to the available range yielding a valid control action $u_k \in \langle u_{\min}; u_{\max}\rangle$ and an updated exploration term Δu_k consistent with the saturated value is computed

$$\Delta u_k = u_k - h(\boldsymbol{x}_k, \boldsymbol{\xi}_k, \boldsymbol{\Psi}_k). \tag{4.48}$$

The updated exploration term is used together with the temporal difference produced by the critic to update the parameters of both actors

$$\boldsymbol{\xi}_{k+1} = \boldsymbol{\xi}_k + \alpha_{\mathrm{ae}}\delta_{k+1}\Delta u_k \nabla_{\boldsymbol{\xi}} \bar{h}(\boldsymbol{x}, \boldsymbol{\xi}, \boldsymbol{\Psi}), \tag{4.49}$$

$$[\boldsymbol{\Psi}_{ij}]_{k+1} = [\boldsymbol{\Psi}_{ij}]_k + \alpha_{\mathrm{ad}}\delta_{k+1}\Delta u_k \nabla_{\boldsymbol{\Psi}_{ij}} \bar{h}(\boldsymbol{x}, \boldsymbol{\xi}, \boldsymbol{\Psi}), \tag{4.50}$$

where α_{ae} and α_{ad} are learning rates of the energy shaping and damping injection actors and $\bar{h}(\boldsymbol{x}, \boldsymbol{\xi}, \boldsymbol{\Psi})$ denotes the control policy saturated to the valid range of control actions. The saturation of the control actions has to be taken into consideration because it zeroes the gradient of the policy when it is out of bounds.

4.11. Conclusions

This chapter provided a brief introduction to the reinforcement learning together with the description of the three different methods that were selected for implementation, simulation and experimental evaluation with the physical magnetic platform. Even

[3]The parameter $\boldsymbol{\Psi}$ is a three-dimensional matrix, so its elements $\boldsymbol{\Psi}_{ij}$ are vectors.

though two of the selected methods use partial (IRL) or even full knowledge (EBAC) of the model of the controlled system, they should be able to adapt to the unmodeled dynamics. All of the three methods seem to be suitable for learning a control policy in real-time along the trajectories of the controlled system. The implementation of the methods and experimental results achieved in simulations and with real magnetic platform are addressed in Chapters 6 and 7.

5. Auto-identification

This chapter addresses the topic of a simple controller design for the magnetic platform that uses auto-identification approach to control differently sized balls. The auto-identification scheme can be seen as an instance of adaptive control. The main difference between the adaptive and learning control is that while the learning control is very flexible and it attempts to find the suitable control strategies itself like the reinforcement learning, the adaptive control rather relies on tuning of parameters of some rigid structure, which already defines the control strategy up to certain extent. However, the adaptation ability still allows dealing with imprecision of the model of the controlled system or reacting to the changes of the system's parameters in real-time. The adaptive control is a well-explored area with a number of publications and overviews available like [28] or [17].

The adaptive control methods can be classified with respect to different criteria. First, we can distinguish between direct (or implicit) and indirect (or explicit) adaptive control. As the term suggests, the direct adaptive schemes adjust directly the parameters of the controller, not attempting to identify any model of the controlled system. On the contrary, the indirect scheme first estimates the parameters of the model of the controlled system and then applies some controller design method to determine the parameters of the controller analytically.

We can also distinguish them by the time horizon of the adaptation. There are methods with *continuous adaptation* ([28]), which never stop estimating and tuning the parameters and therefore are suitable for control of systems whose parameters change over the time. The other option is to adjust the controller parameters during some initialization phase and then continue with the tuned controller as in the traditional feedback control. Such approach is referred to as *vanishing adaptation* ([28]). The vanishing adaptation is suitable for control of imprecisely modeled plants with constant parameters, because it cannot adapt to any changes of the controlled system after the initialization phase. Unlike the continuous adaptation, the vanishing adaptation usually does not have to deal with obstacles like the insufficient excitation of the controlled system in steady state, which is a typical problem of closed-loop identification.

As we have devoted a lot of time and effort to get into the reinforcement learning and the scope of the thesis is limited, we decided to design a simple auto-identification scheme instead of studying and implementing some more sophisticated adaptive method like *iterative feedback tuning* (IFT, [24]), *model-reference adaptive control* (MRAC, [17]) or *multiple-model adaptive control* (MMAC, [17]). Because a pure feedforward control is not suitable for our application, we could also consider some hybrid schemes combining feedforward *iterative learning control* (ILC, [42],[47]) with traditional feedback control into schemes called *feedback error learning* or *current cycle feedback* (FEL, CCF, [42]).

5.1. Suggested approach

In the auto-identification approach, we will exploit the feedback linearization principle described in Section 2.1.1. Although the magnetic force exerted by a magnetic field on a ball scales with the mass of the ball and the acceleration of the ball produced by a given force scales with the inverse of its mass, which seem to compensate each other, the behavior of differently sized balls on the magnetic platform differs. The difference is caused by the fact that the mass of a smaller ball is concentrated closer to the coil array, where the the magnetic field is stronger. However, even though the same coil activation factors produce different accelerations for differently sized balls, for a fixed ball size, the resulting acceleration produced by a coil still scales with the coil activation factor. This fact allows us to parametrize the feedback linearization in such way that we will be able to use the same feedback controller for differently sized balls while maintaining the control performance.

The feedback linearization presented in Section 2.1.1 is parametrized by an identified force profile and therefore computes coil factors needed to produce a required force. This approach would not work for differently sized balls when using the same controller parameters due to the different behavior of the balls described above. However, if we parametrize the feedback linearization by an acceleration profile, the controller will be able to command ball acceleration instead of exerted force, which will allow it to control differently sized balls in the exactly same way. The suggested auto-identification method is based on identification of ball acceleration magnitude profile as a function of distance from a single energized coil. It is sufficient to identify the magnitude of acceleration, because its direction yields from the position of the ball with respect to the activated coil. We will refer to the "acceleration magnitude profile" as "acceleration profile" from now on. The acceleration profiles can be measured for different balls separately and subsequently are used to parametrize the linearization component, so that the ball position controller can always command a required acceleration regardless of the size of the controlled ball. The optimization constraint 2.4 from Section 2.1.1 is modified to

$$\underbrace{\begin{bmatrix} A_{x1,1}(x,y) & \dots & A_{x4,4}(x,y) \\ A_{y1,1}(x,y) & \dots & A_{y4,4}(x,y) \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} U_{1,1} \\ \vdots \\ U_{4,4} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} a_x \\ a_y \end{bmatrix}}_{\mathbf{b}}, \tag{5.1}$$

where the elements $A_{xm,n}(x,y)$ and $A_{ym,n}(x,y)$ are defined with the use of the identified acceleration profile $a(d)$ and the distance d between the ball and a particular coil $[m,n]$ as

$$\begin{aligned} A_{xm,n}(x,y) &= \frac{-a(d)\cdot(x-m)}{d}, \\ A_{ym,n}(x,y) &= \frac{-a(d)\cdot(y-n)}{d}. \end{aligned}$$

The auto-identification approach can be overall classified as an indirect adaptive scheme with vanishing adaptation. We assume that the parameters of the system do not change over time.

5.2. Open-loop acceleration profile identification

During the initialization phase, we use an open-loop coil switching to identify the ball acceleration profile. The identification assumes that the magnetic field produced by every single coil is radially symmetrical and that all of the coils produce the same magnetic field. We do not consider the mutual influencing of the neighboring coils. Although the mutual influencing should be present in reality due to the iron slab connecting the cores of coils on each module, the force measurements mentioned in Section 2.1 have proven that the effect of mutual influencing is negligible.

With the above assumptions, we can freely choose an arbitrary internal coil c_{ij} of the array[1] and use it for the acceleration profile identification. The ball is attracted to one of the coils neighboring with the coil c_{ij} and when it settles in the new position, the neighboring coil is turned off and the coil c_{ij} is fully energized. The oscillations of the ball above the coil c_{ij} are measured and the data is stored for subsequent offline processing. The whole process is repeated several times for all neighbors of c_{ij} to collect a sufficiently large dataset.

Since we are estimating acceleration from a noisy position measurement, it is advantageous to process the data offline, because the offline processing allows using non-causal symmetrical differentiation filters which better suppress the noise and produce more precise results than some higher order backward difference filters that would have to be used for online processing. We compute the acceleration using noise robust differentiators for second derivative estimation from[25]. The differentiators are odd-length symmetrical digital filters and they can compute precise second derivative on up to third order polynomials[2]. The second derivative of a series $\boldsymbol{x}$ at point $x(k)$ is computed as

$$x''(k) \approx \frac{1}{2^{N-3}T^2}\left[s_0 x(k) + \sum_{i=1}^{M} s_i\Big(x(k+i) - x(k-i)\Big)\right] \tag{5.2}$$

where $N \geq 5$ is the odd length of the filter, T is the sampling period of the differentiated signal, $\{s_i\}_{i=0}^{M}$ are the coefficients of the differentiator filter and $M = (N-1)/2$.

[1]By an internal coil of the array, we mean a coil that is surrounded with other coils from all sides.
[2]A completely precise result is achieved only for a noise-free signal.

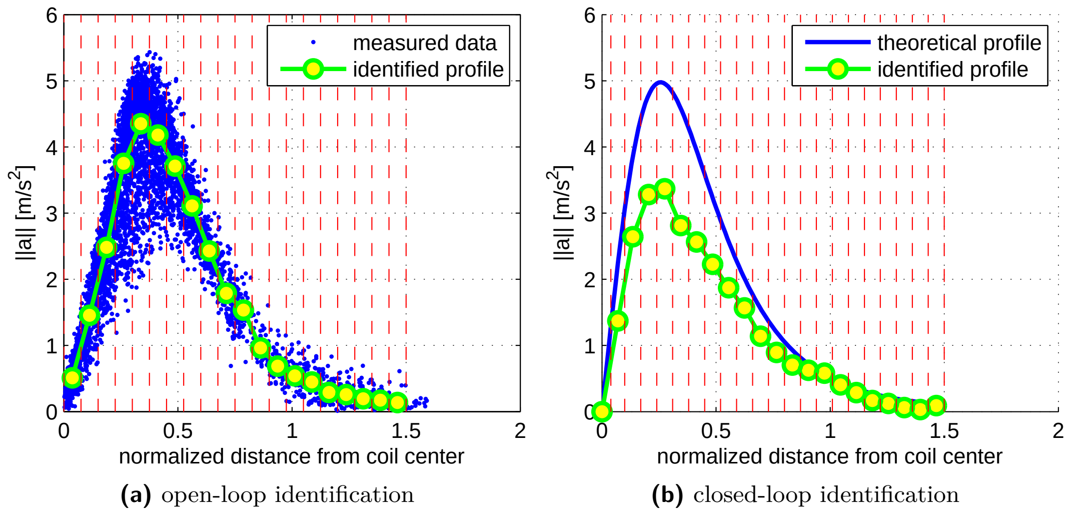


Figure 5.1. Identification of the acceleration profile with a 30 mm ball. Graph (a) shows the result of open-loop identification. The measured data points are marked with blue dots, the red dashed lines show the borders of the bins for acceleration sorting. The yellow points denote the median-filtered values for each bin. This illustrative example uses a lower number of bins than the identification algorithm to improve readability. Graph (b) shows the result of closed-loop identification. The acceleration profile which yields from the analytical force profile described in Section 2.1 is plotted with blue line for comparison, because the data from the closed-loop identification is not suitable for visualization.

The differentiator filter coefficients s_i are generated by a recursive algorithm

$$s_i = \begin{cases} 0 & \text{for } i > M, \\ 1 & \text{for } i = M, \\ \frac{(2N-10)s_{i+1}-(N+2k+3)s_{i+2}}{N-2i-1} & \text{otherwise.} \end{cases} \tag{5.3}$$

The acceleration profile identification has three phases. First, we compute the x– and y–acceleration component from the position in x– and y–direction according to 5.2. The computation is efficiently implemented by convolution of the measured data with the differentiator filter $\boldsymbol{s}_d$ with coefficients 5.3. We determine the magnitudes of the acceleration vector $\boldsymbol{a} = \begin{bmatrix} a_x & a_y \end{bmatrix}^T$ over time

$$||\boldsymbol{a}(k)|| = \sqrt{\boldsymbol{a}_x^2(k) + \boldsymbol{a}_y^2(k)}, \qquad \boldsymbol{a}_x = \boldsymbol{x} * \boldsymbol{s}_d, \;\; \boldsymbol{a}_y = \boldsymbol{y} * \boldsymbol{s}_d \tag{5.4}$$

Second, we sort the acceleration magnitudes into K bins according to the corresponding distance from the center of the activated coil c_{ij} at which the acceleration occurred. The binning of acceleration according to distance is illustrated in Figure 5.1a. Finally, we apply median filtering in each bin to estimate the magnitude of acceleration as a function of distance from the coil center. The function is defined in a tabular form as a set of pairs $\{d_i, \bar{a}_i\}_{i=1}^{K}$ where the values d_i are the distances corresponding to the centers of the bins and $\bar{a}_i$ are the acceleration magnitudes. The pairs are plotted with yellow points in Figure 5.1a. We have to be aware that the binning approach is sensitive to the choice of bin size, which can bias the results, but the sensitivity does not become evident if we use a sufficiently large dataset for the identification.

5.3. Control with identified acceleration profile

After the identification of the acceleration profile, the control scheme is completely analogous to the force feedback linearization approach described in Section 2.1.1, with the only difference that the controller commands acceleration of the ball instead of the exerted magnetic force. The feedback linearization block uses the identified acceleration profile as a lookup table to determine the contribution of each coil to the overall acceleration of the ball at every time step, depending on how far from the coil the ball is. Because the acceleration profile is defined by a set of discrete values and the distance of the ball from a coil is a continuous variable, we use linear interpolation between the discrete values to compute the acceleration contribution for any distance from a particular coil when it is fully energized. The contributions of coils that are not fully energized scale proportionally according to their activation factors.

When the position controller commands acceleration vector $\hat{\boldsymbol{a}}$, the feedback linearization block solves an optimization problem

$$\min ||\boldsymbol{f}|| \text{ s.t. } \hat{\boldsymbol{a}} = \sum_{i=1}^{n} \boldsymbol{v}_i \bar{a}(d_i) f_i, \qquad \boldsymbol{f} = [f_1, f_2, \dots, f_n]^T, \tag{5.5}$$

where n is the total number of coils, $\boldsymbol{v}_i$ is unit direction vector pointing from the ball position towards the i–th coil, $\bar{a}(d_i)$ is the interpolated acceleration magnitude for fully energized i–th coil and f_i is its activation factor, which has to be determined. The activation factors are afterwards converted to PWM duty cycles that produce the commanded acceleration of the ball.

5.4. Closed-loop acceleration profile identification

The acceleration profile can be also identified under closed-loop control, which brings a possibility to periodically improve the estimate of the acceleration profile while simultaneously controlling the ball. We can collect the information about the position of the ball and coil activation factors over the time in the closed-loop. Having a sufficient amount of such data from N time-steps, we can turn the constraint from the optimization problem 5.5 into a least-squares problem where the vector of acceleration magnitudes $\bar{\boldsymbol{a}}$ is unknown, using the binning approach from Section 5.2 with K equidistant bins centered at positions d_i

$$\boldsymbol{\Gamma}\bar{\boldsymbol{a}} = \boldsymbol{z}, \qquad \boldsymbol{\Gamma} \in \mathbb{R}^{2N\times K}, \bar{\boldsymbol{a}} \in \mathbb{R}^{K}, \boldsymbol{z} \in \mathbb{R}^{2N} \tag{5.6}$$

The right-hand side $\boldsymbol{z}$ is formed by stacking the observed ball accelerations over time as $\boldsymbol{z} = [a_{x1}, a_{y1}, a_{x2}, a_{y2}, \dots, a_{xN}, a_{yN}]^T$. The formation of matrix $\boldsymbol{\Gamma}$ is a bit more complicated. The data from each time-step are used to generate two rows of the matrix according the Algorithm 5.1. The resulting least-squares problem for

Algorithm 5.1 Building least-squares problem for closed-loop identification

```
inputs from $k$–th time-step: position $[x, y]^T$, coil activation factors $\boldsymbol{f}$
initialize rows $\boldsymbol{\Gamma}_{2k}$ and $\boldsymbol{\Gamma}_{2k+1}$ with zeros
compute direction vectors $\boldsymbol{v}_i$ and distances $d_i$ from the position $[x, y]^T$ to all $n$ coils
**for** $i = 1 \dots n$ **do**
    $b \leftarrow$ [the bin index corresponding to distance $d_i$]
    $\boldsymbol{\Gamma}(2k, b) = \boldsymbol{\Gamma}(2k, b) + [v_x]_i f_i$
    $\boldsymbol{\Gamma}(2k+1, b) = \boldsymbol{\Gamma}(2k+1, b) + [v_y]_i f_i$
**end for**
```

acceleration magnitudes is afterwards solved in a standard way

$$\bar{\boldsymbol{a}} = \left(\boldsymbol{\Gamma}^T\boldsymbol{\Gamma}\right)^{-1}\boldsymbol{\Gamma}^T\boldsymbol{z}. \tag{5.7}$$

The closed-loop identification needs sufficient excitation of the controlled system which requires a rich reference signal. We chose a combination of harmonic and stair-like reference signals to force some faster transients, because it is impossible to extract useful information from the closed-loop in steady state or during slow changes. If we compare Figures 5.1a and 5.1b, it is obvious that even with a rich reference signal, the closed-loop identification is outperformed by the open-loop variant. The open-loop identification is more precise because the ball under the closed-loop control usually does not reach as high acceleration as during the free oscillations above a fully energized coil. The low acceleration generated by the coils that are not fully energized can be less noticeable than the measurement noise. In addition, the closed-loop identification may also suffer from the correlation of the control input with the measurement noise, which is a common problem of closed-loop identification.

5.5. Conclusions

We used the existing mathematical model and feedback linearization approach to design a simple adaptive control scheme based on auto-identification procedure, which allows controlling position of differently sized balls using the same ball position controller while maintaining the control performance thanks to the parametrization of the underlying feedback linearization. The evaluation of the suggested approach in simulations and with the real platform is given in Chapters 6 and 7. The methods and models for collecting the data for auto-identification can be used for enhancement and further analysis of the mathematical model of the platform in the future, e.g. to determine whether the damping of the movement of the ball is mainly caused by the rolling resistance on the surface of the platform, magnetic hysteresis or eddy currents in the body of the ball.

6. Simulations

This chapter describes the implementation details and the results achieved with the particular reinforcement learning methods and auto-identification in simulations. The methods were first implemented as pure simulations in MATLAB functions and scripts, because such implementation makes tuning and debugging of the algorithms quite efficient. The simulations also allow to avoid all possible real-world complications like measurement noise, disturbances, actuator saturations, delays or model imprecisions that could cause problems when deploying the algorithms with the real magnetic platform. It is therefore reasonable to carry out the simulations first to find out whether the problem is feasible at all. We started evaluating the methods on a noise-free linear system, gradually adding the noise, saturations etc. to the model. After being simulated, the control algorithms were implemented in Simulink and evaluated with the real platform.

The first step with the reinforcement learning methods is the control of ball position in one dimension, let's say along the x–axis, above a single row of coils. The chosen control objective for evaluation of reinforcement learning is to stabilize the ball in the middle of the row of coils. In the presented results, we use the normalized platform coordinates defined in Section 2.1, but we add an offset, so that the origin of the coordinate system corresponds to the center of the platform.

The position control in two dimensions was planned as an extension after fully mastering the one-dimensional control, including the set-point tracking and command following. However, even the one-dimensional stabilization appeared to be quite a complicated task, so the two-dimensional control is not addressed. We agreed with the thesis supervisor to focus on the deployment of the control algorithms for a single ball and leave the parallel manipulation mentioned in the assignment of the thesis as a future work.

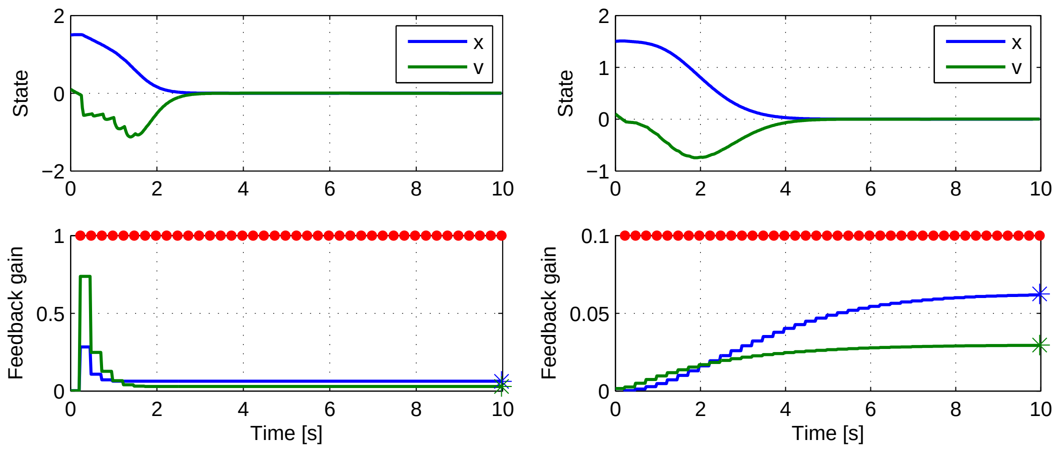
6.1. Online IRL

The IRL requires full state information and knowledge of the system's input gain, i.e. the projection from the inputs to the states. The state provided to the IRL does not necessarily have to be directly the position and velocity of the ball, but it has to be possible to reconstruct them from the provided information. This is a relatively strict limitation of the IRL, because designer needs to have a good knowledge of the structure of the controlled system in order to decide which measurements to use. The benefit of the IRL is that it can adapt to unknown parameters of the dynamics of the system. Although the ball one-dimensional movement dynamics is modeled as a double integrator (see Equation 2.1), we have already mentioned the presence of some damping. The damping is not modeled, quantified or explored, so the actual state transition matrix $\boldsymbol{A}$ is perturbed, which is the area where the IRL can be helpful.

6.1.1. Noise-free system

The policy iteration and value iteration from Section 4.8 with batch least-squares weight estimation were initially implemented with a noise-free model. We used a model of a double integrator both with and without damping to generate data for the learning algorithms, which only knew the data and the input gain of the model, but they did not know that the model was a double integrator neither if it was damped or not. The simulations have confirmed that both of the algorithms converge to the optimal solution of the LQ–problem. The convergence of the policy and value iteration along a trajectory for a noise-free double integrator is illustrated in Figure 6.1. However, although we obtained convergence, two interesting issues occurred.

First, the least-squares estimation of the Riccati solution $\boldsymbol{P}$ requires persistent excitation (PE, [6]) of the system throughout the estimation process. The PE is a sufficient condition for convergence of the least-squares estimation. For a least-squares problem $\boldsymbol{\Gamma\theta} = \boldsymbol{z}$, the PE requires that the data accumulated in matrix $\boldsymbol{\Gamma}$ at



(a) policy iteration **(b)** value iteration

Figure 6.1. Convergence of the IRL algorithms for a noise-free double integrator, learning on batches of 10 samples. The optimal feedback gains computed just for comparison by solving the corresponding CARE are marked with asterisks at the ends of gain transients. The feedback gains are plotted with the same color as the corresponding state. The red dots indicate satisfying the PE for every batch, but the PE is only theoretical, because the amplitudes of the states at the end of the transients are in order around 10^{-12}, which is practically zero. It is obvious that the policy iteration at (a) converges significantly faster than the value iteration at (b).

the moment of estimation of parameter vector $\boldsymbol{\theta}$ satisfy condition

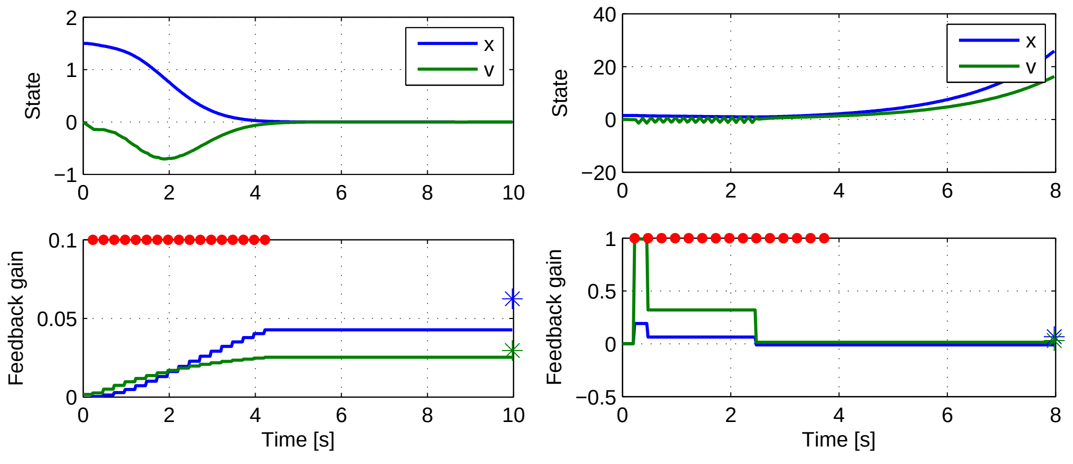
$$c_1\boldsymbol{I} < \sum_{i=1}^{\mathrm{N}} \boldsymbol{\Gamma}_i\boldsymbol{\Gamma}_i^T < c_2\boldsymbol{I} < \infty, \qquad c_1, c_2 > 0, \tag{6.1}$$

where $\boldsymbol{\Gamma}_i$ is the vector of data collected at the i–th step of the estimation horizon $i = 1 \dots \mathrm{N}$. Naturally, the length of the estimation horizon has to be greater than or equal to the number of the estimated parameters. Both policy and value iteration of the IRL attempt to converge to the optimal feedback gain online along the trajectory of the controlled system, so the states are being regulated to the origin during the learning process, which can destroy the PE before the algorithm converges.The authors of [31] suggest injecting a probing noise into the control input to ensure PE, but such solution is suitable just for gradient descent versions of the algorithms. We have tried the probing noise injection with the least-squares parameter estimation and we observed that the probing noise completely biased the parameter estimation, which destroyed the convergence of the parameters to the optimal values. Another option of ensuring the PE is to reset the system to a new (possibly random) initial state when the PE is lost. This solution was also used by the authors of [31], although not explicitly mentioned in the text. The state resetting preserves the convergence of the algorithms, but it can cause difficulties for a real system. If we are unable to drive the system to a required initial state, we can still generate a new initial state by applying a sequence of random actions. However, then we have to be sure that the random actions cannot damage the controlled system. In our case, the ball could e.g. get out of the area of attraction of the coil array. Beside checking the PE condition 6.1, we also check the condition number[1] of the matrix $\boldsymbol{\Gamma}$ to prevent solving an ill-conditioned problem.

The second issue, also related to the problem of PE, is the sensitivity of the algorithms to the tuning parameters, namely to the weighting matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ of the LQ–criterion and the initial control policy. It is not surprising that the states of the system can be regulated close to the origin before the gains converge to the optimal solution, especially when the weighting matrices prefer state penalization to control input penalization to get a more aggressive controller. Although the Figure 6.1 shows that the PE is accomplished throughout the whole simulation, it is satisfied only theoretically, because the amplitudes of state variables are below the order of 10^{-6} after 6 seconds. Nevertheless, we can see that the value iteration algorithm converges until the end of the simulation. Due to some measurement noise, disturbances etc., it is impossible to estimate any parameters from such tiny values in a practical application. If we impose a lower bound on the state amplitude for a policy update, the policy iteration still converges fast enough, but the value iteration is too slow to be learned from a single transient, as shown in Figure 6.2a.

The performance of policy iteration can be also affected by the choice of the initial stabilizing gain. Figure 6.1a shows an initial overshoot of feedback gains. Our empirical experience is that the overshoot grows with the distance of the initial feedback gain

[1]The condition number of a matrix is the ratio of its largest and smallest singular value. A high condition number indicates an ill-conditioned least-squares problem.



(a) slow convergence of value iteration **(b)** instability of policy iteration

Figure 6.2. Issues of the IRL algorithms for a noise-free double integrator. Figure (a) shows too slow convergence of the value iteration when we impose a lower bound on the state amplitude used for update. Figure (b) illustrates instability of the closed-loop system with a control delay under policy iteration control. The other parameters (weighting matrices, initial control policy, batch length) are the same as in Figure 6.1.

vector from the optimal one. The high initial feedback gain can cause instability for a real system when there is a delay in the control loop caused by state measurement, computation and propagation of the control action. Figure 6.2b shows the instability of the policy iteration learning process caused by a control delay. Although the learning parameters were the same as for the example at Figure 6.1a, the control delay causes initial oscillations that result in bad parameter estimation and destabilizing feedback gains. We have also experienced situations when the high initial feedback gains drove the system on a trajectory which headed towards the origin, but required so little control action that it led to loss of PE after the first update of the control policy even in the simulation.

6.1.2. Trial learning

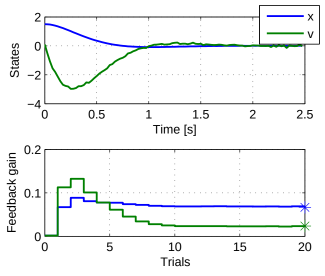
This section addresses the usage of IRL with a noisy state measurement. We consider an additive noise with normal distribution and zero mean value and vary its standard deviation to inspect its influence on learning. The usual period of policy updates for the IRL is tens of samples collected along a trajectory ([31]). However, when we introduce a measurement noise, such batches of noisy data are not long enough to give a good estimate of nominal parameter values and the learning destabilizes the closed-loop system. The problem is that the batches of data for the least-squares problem are built from scratch after every policy update, so that the previously accumulated information is lost. We have also unsuccessfully tried the recursive variant of least-squares (RLS) with limited exponential forgetting, which would retain the past information. As the RLS needs to be initialized with large values in the covariance matrix of the parameters to adapt well, the initial estimates are strongly biased by the noise and yield a destabilizing feedback gain, from which the algorithm does not recover.

We therefore modified the batch learning to a trial learning, updating the policy only after completion of a trial. The length of the trials is chosen so that it is possible to regulate the system within a single trial and after each trial, the system is reset to a new initial state to ensure excitation. Overall, the trials are equivalent to longer batches, allowing to observe several hundreds of samples for a more reliable update. The extension of learning batches can suppress the influence of the measurement noise up to a certain extent.

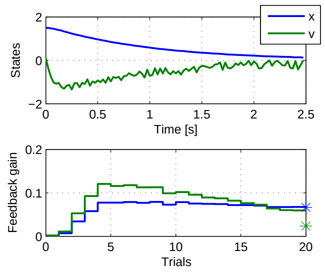
Under the presence of noise, we found it reasonable to explicitly force the negative definiteness of the estimated CARE solution $\boldsymbol{P}$ which yields from the parameter vector $\boldsymbol{\theta}$. When the update does not yield a negative definite estimate of $\boldsymbol{P}$, it is rejected and the trial data is disposed, which slows down the learning, but prevents the destabilization of the closed-loop. We have to emphasize that a negative definite estimate of $\boldsymbol{P}$ does not guarantee that the resulting feedback gain is stabilizing, but it can filter at least some of the bad cases. On the other hand, with a growing noise level, more updates are being rejected. The update rejection causes that sometimes the algorithms perform many trials without a single update and when the noise is too strong, they do not update at all.

The simulations have shown that the value iteration is less susceptible to noise than the policy iteration, which can be explained by the fact that it updates the control policy in smaller steps, resulting in more trials needed for learning. Behavior of both algorithms with different levels of noise is compared in Figures 6.3 and 6.4. With a low noise level, the trials are sufficiently long to get convergence to the optimal feedback gain values. When the noise level grows, the policy iteration converges to biased values instead of the optimal ones (Figure 6.3b), while the value iteration retains performance. If we increase the noise level even more, many policy updates are rejected and while the value iteration converges to biased values after the initial update rejections, the policy iteration gets unstable due to a bad policy update, which was not captured by the $\boldsymbol{P}$ negative definiteness check . It is noticeable that the policy iteration turns to a less aggressive control strategy with the growing level of noise.

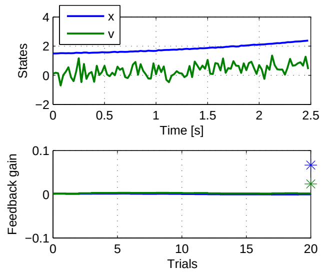
Even though the learning in trials is more resistant to measurement noise than the original IRL, its resistance is limited. The noise level of the foil position measurement is too high to be dealt with the IRL policy iteration, while the camera measurement is less noisy, but it has a significant delay, which destroys convergence. Only the value iteration is a reasonable candidate for the real platform. As we are directly measuring only the position of the ball and the velocity has to be determined using differentiation, the noise is amplified. The noise amplification is the reason why the standard deviations for velocity in Figure 6.3 are approximately 40 times higher than the standard deviations for position. Furthermore, not only the learning, but also the feedback linearization will suffer from imprecise position measurement, which will



(a) $\sigma_x = 0.001$, $\sigma_v = 0.04$

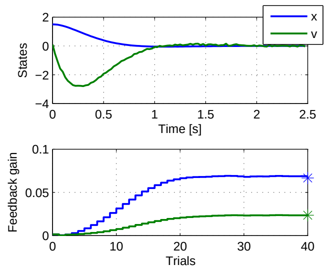


(b) $\sigma_x = 0.003$, $\sigma_v = 0.12$

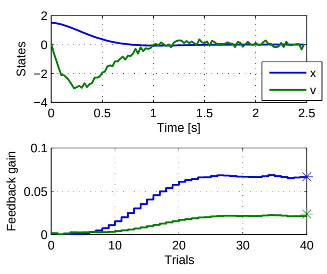


(c) $\sigma_x = 0.01$, $\sigma_v = 0.4$

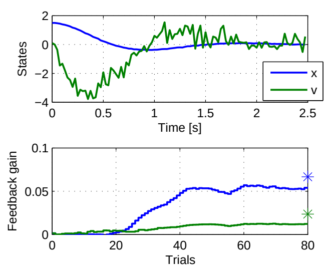
Figure 6.3. IRL policy iteration performance with different levels of measurement noise. The lower figures show the transients of feedback gains averaged over 20 runs. The upper figures show a sample trial trajectory from the end of the learning process for different noise levels. The feedback gains are plotted with the same color as the corresponding state, with the theoretical optimal values marked with asterisks.



(a) $\sigma_x = 0.001$, $\sigma_v = 0.04$



(b) $\sigma_x = 0.003$, $\sigma_v = 0.12$



(c) $\sigma_x = 0.01$, $\sigma_v = 0.4$

Figure 6.4. IRL value iteration performance with different levels of measurement noise with the same parameters as for policy iteration presented in Figure 6.3. Note that the number of learning trials was doubled for the highest noise level in (c).

lead to a bigger difference between the commanded and exerted magnetic force. The experimental results with real platform are presented in Section 7.

6.2. Online LSPI bang-off-bang

Similarly to the IRL, the evaluated control objective for the LSPI was the stabilization of the ball in one dimension in the center of a single row of coils. The evaluation of the online LSPI included selecting suitable parameters for the BFA for Q-function approximation. The BFA uses Gaussian radial basis functions (GRBF) that are distributed in a grid above the two-dimensional phase plane of the double integrator. The grid is equidistant in each dimension separately and the grid step determines the variance of the GRBF in the respective dimension. The value of a GRBF at position $[i;j]$ of the grid is defined as

$$\phi_{ij}(x,v) = k e^{-\left[\frac{(x-\mu_i)^2}{2d_x^2} + \frac{(v-\mu_j)^2}{2d_v^2}\right]} \tag{6.2}$$

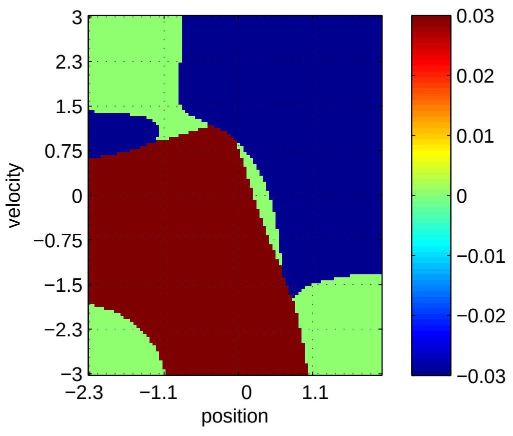
where x, v are the position and velocity, μ_1, μ_2 are the coordinates of the GRBF in the phase plane and d_x, d_v are the grid steps for position and velocity. The normalization constant k is variable, because the whole vector of basis function values is normalized to have a unit sum at every step.

For the double integrator control, we have cut down the size of the GRBF grid to 3×3, which appeared to be sufficient to learn a Q-function that yields a control policy for stabilization of the ball in the middle of a row of coils. We have 9 unique basis functions that are replicated for each available control action, so the basis function vector contains 27 functions in total. The unique basis functions are located at phase plane positions $\boldsymbol{\mu} = \{-1.5, 0, 1.5\} \times \{-2, 0, 2\}$, both position and velocity expressed in the normalized platform units. The position coordinate is offset to get $x = 0$ in the middle of the row of coils, so $x_1 = \pm 1.5$ correspond to the centers of the outer coils of the row. The range for the velocity was chosen to cover the range of speeds reached by the ball during the learning process. The set of control actions was selected as $U = \{-0.03, 0, 0.03\}$ to balance between fast and smooth transients. The learning process is split into 30 trials that start from a fixed initial state and last for 5 seconds. The updates of the BFA are performed several times per trial with exponentially decaying exploration rate, using a quadratic reward function $r(\boldsymbol{x}_k, u_k) = \boldsymbol{x}_k^T \boldsymbol{Q} \boldsymbol{x}_k + R u_k^2$ with negative definite $\boldsymbol{Q}$ and R. Due to the exponential exploration decay, there is a possibility that the algorithm will not converge to a stabilizing greedy control policy because of bad initial exploration. When the policy does not approach to a stabilizing one before the exploration rate drops significantly, the algorithm will stay stuck with a bad policy.

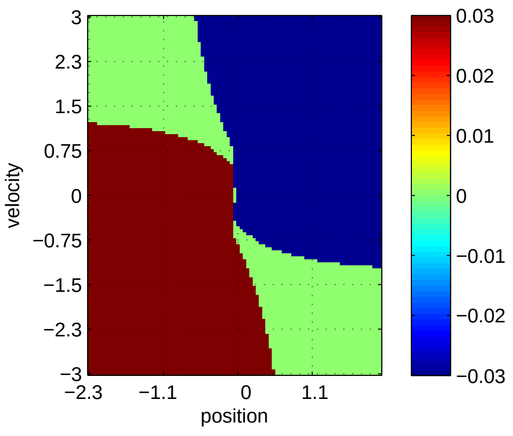
The simulations have shown that the algorithm can converge quite fast even under the presence of noise, because the GRBF have a smoothing effect on the noisy data. However, we were not able to reach zero steady-state error with the resulting greedy policy. An example of a greedy control policy with equilibrium offset is shown in Figure 6.5a. We can clearly see that the zero-control area between the positive and negative control action regions does not intersect the line $x_2 = 0$ in the phase plane at the origin, while the resulting equilibrium is located within the intersection area, as shown on the sample trajectory in Figure 6.6a controlled by the greedy policy in Figure 6.5a. The offset is caused by asymmetrical distribution of the exploration in the phase plane during the learning process, which results in different precision of Q-function estimation in areas corresponding to different GRBF.

6.2.1. Symmetry exploitation

We decided to exploit the symmetry of the double integrator to prevent the offset of the learned equilibrium. We know that the control policy for a double integrator should be antisymmetric with respect to the origin of the phase plane, so we learn the Q-function only in the right half of the phase plane where $x_1 \geq 0$, using the point antisymmetry to generate Q-function values in the other half of the phase plane when needed. Beside decreasing the number of learned parameters from 27 to 18, this modification avoids the offset of the resulting equilibrium, as shown in Figure 6.5b.



(a) original LSPI

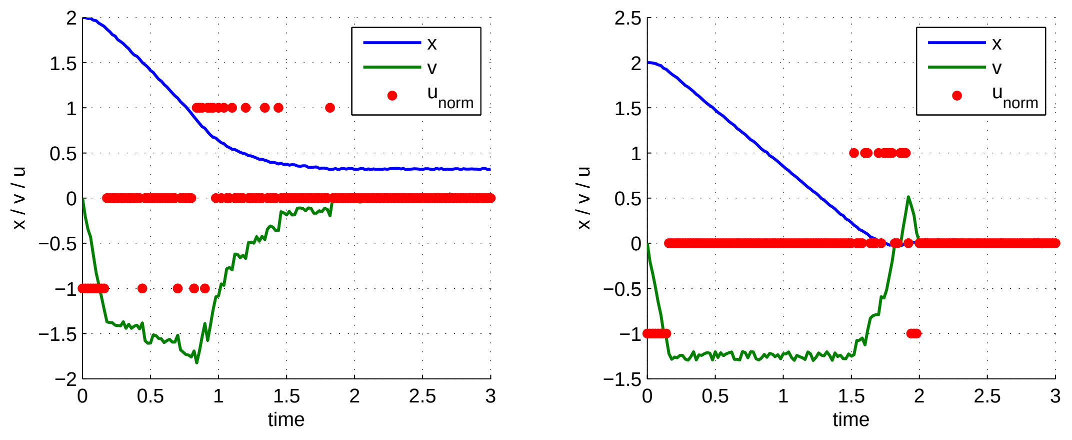


(b) symmetry-exploiting LSPI

Figure 6.5. Examples of greedy control policies after the LSPI learning process visualized in the phase plane. Red color denotes $u = u_{\max}$, blue color $u = u_{\min}$ and green color $u = 0$. The policy (a) has a visible positive offset of the equilibrium position. The small green area around the origin in symmetry-exploiting policy (b) is an explicitly defined zero-control region which prevents oscillations around the equilibrium.

Unfortunately, the exploration of the phase plane is still not symmetrical, so we can notice that the learned and generated part do not connect perfectly smoothly and there is a very sharp changeover between the positive and negative control action regions around the origin. Such sharp changeover in practice causes small steady oscillations around the origin, which is a common problem of switching control that can be solved by explicitly defining a zero-control area around the equilibrium. The zero-control area introduces a small steady-state error, but the oscillations are avoided. The symmetry-exploiting LSPI is able to learn faster, because it has to learn less parameters than than the original variant. While we have used 30 trials for the basic LSPI, the symmetry-exploiting LSPI is learned in 15 trials. The example policies in Figure 6.5 show quite good generalization ability of the function approximator, because the trials were initialized at position $\boldsymbol{x}_0 = [1.5, 0]^T$ and the resulting policies are stabilizing further from the origin, too. Examples of stabilization transients are given in Figure 6.6. If we compare the two transients 6.6a and 6.6b, we find out that the original LSPI converged to a more aggressive policy with nonzero steady error, while the symmetry-exploiting LSPI tries to keep the position very close to the origin, so that it sometimes reacts to the measurement noise.

It is noticeable that the policy in Figure 6.5a has areas where it does not stabilize the ball (the lower left corner of the phase plane). However, these states are not visited during learning neither during the subsequent control. It nicely illustrates how the reinforcement learning focuses on the experienced situations. Although it is possible to get a stabilizing policy for the whole region shown here, the learning sacrifices the performance in the rare states to a better performance in the frequently visited ones. Nevertheless, the random exploration sometimes also leads to a control



(a) original LSPI **(b)** symmetry-exploiting LSPI

Figure 6.6. Sample trajectories showing stabilization of the ball by the greedy control policies presented in Figure 6.5. The control input was normalized by $u_{\max}$ to make its amplitude visible compared to the states. Figure (a) shows a remarkable offset of the equilibrium with the original LSPI control, which is eliminated by the symmetry exploitation in Figure (b).

policy that is stabilizing in the whole region, as shown in Figure 6.5b. Overall, the symmetry-exploiting LSPI is a suitable candidate for the real platform.

6.3. Online EBAC

The general MATLAB implementation of the EBAC learning framework was kindly provided by Subramanya Nageshrao from TU Delft, together with a demo example for an inverted pendulum swing-up learning, so we did not have to implement the algorithms from scratch, but accommodated the existing code for our purposes.

For the experiments, we first need to convert the original linear model of the rolling ball (see Equation 2.1) to the port-Hamiltonian form. We change the original position-velocity state vector $\boldsymbol{x}_{\text{orig}} = [x, v]^T$ to a state vector $\boldsymbol{x} = [x, p]^T$, where x is still the position, but p is the magnitude of momentum. It would be possible to use the original state vector as well, but we wanted to be consistent with the existing EBAC framework implementation. Then, we describe the total accumulated energy of the ball in terms of the new state vector. As the platform's surface is horizontal, the accumulated energy of the ball has only the kinetic component and therefore does not depend on the position of the ball. The Hamiltonian function and its gradient are defined as

$$H(\boldsymbol{x}) = \frac{1}{2\mathrm{m}_{\mathrm{ef}}} p^2 \qquad \rightarrow \qquad \nabla_{\boldsymbol{x}} H(\boldsymbol{x}) = \begin{bmatrix} 0 \\ \frac{p}{\mathrm{m}_{\mathrm{ef}}} \end{bmatrix}. \tag{6.3}$$

The resulting port-Hamiltonian representation of the ball rolling in one dimension is

$$\begin{aligned} \dot{\boldsymbol{x}} &= \left\{ \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \right\} \begin{bmatrix} 0 \\ \frac{p}{\mathrm{m_{ef}}} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \\ y &= \frac{p}{\mathrm{m_{ef}}}, \end{aligned} \tag{6.4}$$

where the input port u represents the force exerted by the magnetic field and the output port y is the velocity of the ball, satisfying the port-Hamiltonian description requirement that the product of the conjugated input and output ports is power. Note that the model 6.4 is in standard metric units, which is necessary for expressing the energy of the system. However, the presented simulation results will be scaled to the normalized platform units to make them better comparable to the results achieved with the other methods.

In the next step, we determined which states allow the energy shaping according to the condition 4.41. With the parameters of port-Hamiltonian representation 6.4, the condition 4.41 from Section 4.10.2 turns into

$$\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x} H_{\mathrm{a}}(\boldsymbol{x}) \\ \frac{\partial}{\partial p} H_{\mathrm{a}}(\boldsymbol{x}) \end{bmatrix} = 0, \tag{6.5}$$

which yields that the energy can be shaped only on the position. Therefore the energy shaping actor acts only on the first state variable. The BFA for the critic and the actors use 3rd order harmonic cosine basis, whose advantage is that it automatically ensures the symmetry of approximated value function and policy. On the other hand, the BFA with harmonic basis does not have a generalization ability outside the fundamental period of its basis functions, because the approximation is periodic. The fundamental periods therefore has to be chosen carefully to cover the whole range of the state space, where we want to control the system. The basis function value computation for a single state variable (for the energy-shaping actor) is straightforward

$$\phi_i(x) = \cos(\pi h_i x), \qquad i = 1 \dots 4, \;\; h_i = 0 \dots 3. \tag{6.6}$$

The situation is more complicated when the BFA acts on the whole state vector. The i-th harmonic basis function value acting on a vector $\boldsymbol{x} \in \mathbb{R}^2$ is determined as

$$\phi_i(\boldsymbol{x}) = \cos\left(\pi \boldsymbol{h}_i^T \boldsymbol{x}\right), \qquad i = 1 \dots 16, \tag{6.7}$$

where the vectors $\boldsymbol{h}_i = [h_i^x, h_i^p]^T$ exhaust all possible ordered combinations of harmonic frequencies $h^x, h^p = 0 \dots 3$. The base learning rates for the critic and the actors are normalized by the magnitude of the harmonic frequency vector, but preventing normalization by zero

$$\alpha_i = \frac{\alpha_{\mathrm{base}}}{\max\left\{ ||\boldsymbol{h}_i||, 1 \right\}} \tag{6.8}$$

resulting in vectors of learning rates, which are used for BFA updates. The learning rate normalization is necessary for successful learning, because it forces the approximation to estimate the lower frequencies first and then use the higher ones to approximate the details of the approximated function. Similarly to the online LSPI, the learning relies on a quadratic reward function $r(\boldsymbol{x}_k, u_k) = \boldsymbol{x}_k^T \boldsymbol{Q} \boldsymbol{x}_k + Ru_k^2$.

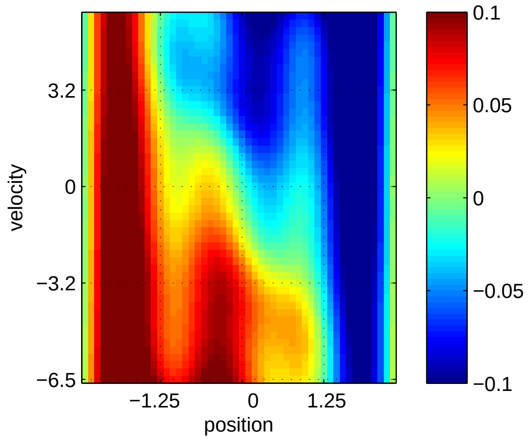
6.3.1. Tuning of the learning parameters

Although the algorithm uses full knowledge of the controlled system, it turned out that it is necessary to tune the base learning rates for the critic and actors and the weighting matrices of the quadratic reward function carefully to get convergence even in the noise-free simulations. The tuning of the learning parameters is a lengthy manual work done rather in a trial-error manner. Although there exist some rules of thumb, like that the critic base learning rate should be higher than the base learning rates for actors, it is not possible to determine the suitable values of the parameters in advance. In addition, as the algorithm uses gradient-based tuning of the parameters, the learning rates are interconnected with the weighting matrices of the quadratic reward function. If we e.g. multiply both of the weighting matrices by a positive constant, thus changing the magnitude of the resulting value function, not its shape, we will probably also have to adjust the base learning rates to preserve convergence, because the change will project into the size temporal difference error which can affect the stability of learning.

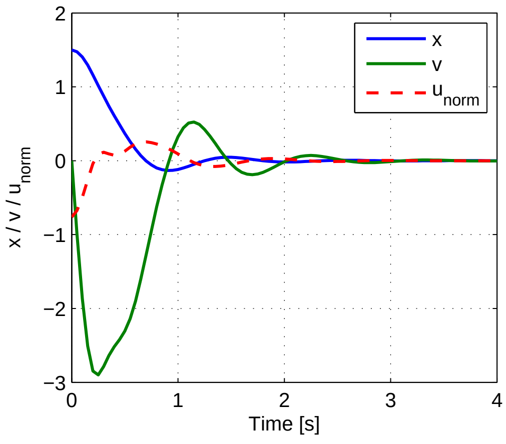
Finally, we succeeded in finding suitable learning parameter values for convergent learning by running a vast number of learning simulations with automatically generated combinations of the base learning rates, checking the resulting cumulative reward at the end of every situation. We attempted to further optimize the selected learning parameter values using the *fminsearch* function from MATLAB's Optimization Toolbox, using our selection as the initial point for search. However, the dependence of the resulting control performance on the learning parameters is so bumpy that the optimization did not improve anything.

Although the algorithm is usually able to converge to a stabilizing policy during the first 15 trials, we use 30 trials in total, each lasting 6 seconds, to fine-tune the policy and get a better control performance. Figure 6.7 shows an example of a learned control policy in the phase plane together with a sample controlled trajectory. It is evident that a major part of the policy presented in Figure 6.7a hits the saturation boundaries of the control input, which were set to prevent the controller from commanding a force that cannot be exerted by the real coil array.

When introducing the measurement noise, we decreased the base learning rates for the actors and critic to 50% of the values that were used for the noise-free simulations to preserve the convergence. The convergence is therefore slower and we use 50 trials for learning, but the algorithm finally arrives to a very similar policy as in the noise-free simulation and successfully stabilizes the ball in the origin, as shown in Figure 6.8.

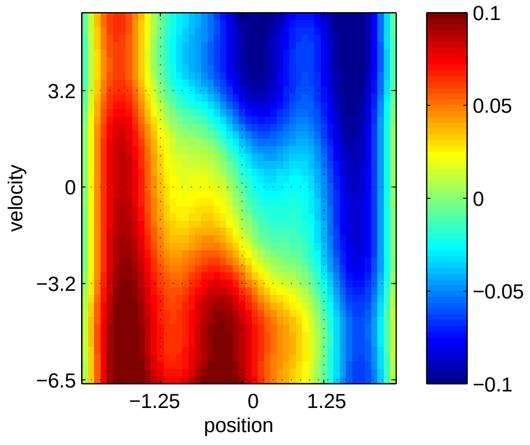


(a) sample control policy

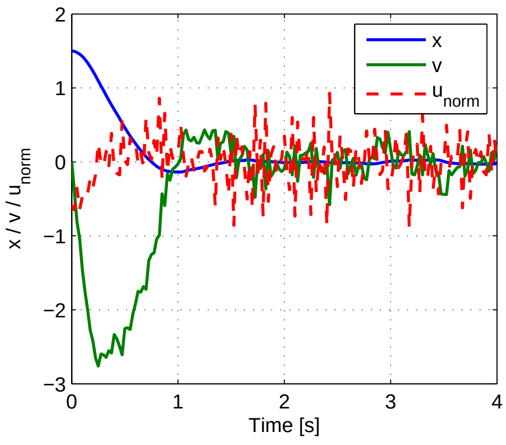


(b) sample controlled trajectory

Figure 6.7. Example of a control policy learned by the online EBAC (a) with a sample trajectory obtained by control of the double integrator system using the policy (b). The control input in (b) has been normalized by u_{max} to make its amplitude comparable to the states.



(a) sample control policy

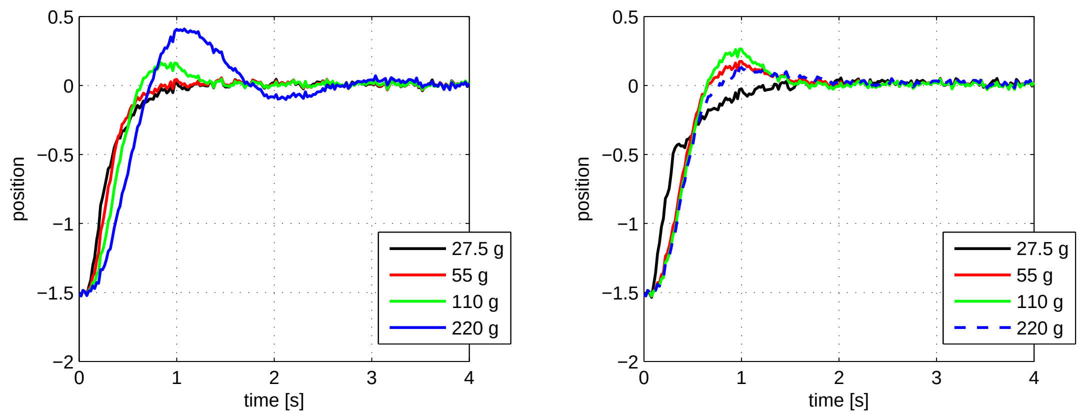


(b) sample controlled trajectory

Figure 6.8. Example of a control policy (a) learned by the online EBAC under the presence of measurement noise with a sample trajectory (b) obtained by controlling the double integrator system using the policy. The control action has been normalized by u_{max} for readability.

6.4. Auto-identification

For the simulation of auto-identification adaptive control, we consider four balls of different sizes. The different size of a ball projects into the change of the input gain of its linear model. In addition, as mentioned in Section 5.1, the resulting magnetic force exerted on a differently sized ball will change as well due to different distance of the ball's center of mass from the coils. However, all of these changes project into



(a) non-adaptive control **(b)** adaptive control

Figure 6.9. The effect of parametrization of feedback linearization with the auto-identified acceleration profiles. All of the simulations were run with the same PIDf feedback controller, the transients in Figure (a) were produced with the use of the manually identified analytical force/acceleration profile for the 110 g ball. The transients in Figure (b) use the identified acceleration profiles.

the acceleration profile, so if we identify the acceleration profile for a particular ball, we can avoid the undesirable effect of its different size on the control performance.

We have used four linear models of the rolling ball corresponding to masses 27.5 g, 55 g, 110 g and 220 g together with the known manually identified model of the magnetic force for the 110 g (30 mm) ball presented in Section 2.1 to generate free-oscillation data as described in Section 5.2. Therefore we have neglected the changes in the magnitude of the exerted magnetic force for different balls in simulations, because we do not have the force profiles for the other ball sizes available. However, any scaling of the magnetic force would project into the measured acceleration profile and therefore would be captured and canceled out during the feedback linearization. Furthermore, the binning approach that we use for the acceleration profile approximation is non-parametric and can possibly adapt even to significant changes of shape of the force profile. The generated data which include a model of Gaussian measurement noise is processed with noise-robust numerical differentiation, resulting in four different acceleration profiles.

For the evaluation purposes, we have tuned a simple PIDf feedback controller to control the 110 g ball with the manually identified analytical model of the magnetic force[2] that was presented in Section 2.1. The PIDf controller was afterwards used to control the differently sized balls from position $x = -1.5$ to position $x = 0$ in the offset normalized coordinates with a step reference input. Figure 6.9 compares the transients for differently sized balls with and without the use of auto-identification. The transients in Figure 6.9a were generated when controlling all of the balls using the analytical model of the magnetic force for feedback linearization, i.e. as if we

[2]The model of magnetic force was converted to a model of acceleration profile using the known effective mass of the 110 g ball to fit into the acceleration profile framework.

were controlling the 110 g ball in all four cases. Even though the controller is able to drive the ball to the reference position, the quality of control differs significantly. The transients in Figure 6.9b were generated with the same feedback controller, but using the corresponding identified acceleration profile to parametrize the feedback linearization for every particular ball. The random generators for the measurement noise were initialized with the same seed every time. We can see that the control performance is preserved in the second case, except for the smallest ball. The transient for the smallest ball differs from the rest because the sampling period used for the identification of its acceleration profile was not sufficiently high and biased the estimated profile. The bigger balls did not oscillate so fast during identification experiment, so their acceleration profiles were estimated more precisely. Increasing the sampling frequency for the identification experiment would solve the problem for the smallest ball, but we wanted to demonstrate the limitations of the method here. The minor differences in the transients for the bigger balls are caused by imprecision of the numerical differentiation and acceleration profile approximation.

6.5. Conclusions

We have implemented simulations for evaluation of the selected control methods to verify their sensibility for a real application. All of the methods work well in ideal noise-free simulations, except for the original LSPI, which suffers from the offset of the equilibrium. We have suggested modifications for the IRL (trial learning to reach convergence with noisy data) and the LSPI (symmetry exploitation for equilibrium offset avoidance and faster learning). Without the modifications, the IRL could not be deployed on a system with measurement noise, and the LSPI would probably result in control policies with offset equilibrium. The EBAC learning framework is able to find good control policies, but it requires manual tuning of the learning parameters, which is lengthy and it would take even longer time when involving a real system instead of performing simulations. The auto-identification approach has been shown to work quite well in simulations, but there is a possibility that we could be limited by the upper bound of the achievable sampling frequency of the position measurement on the real system. After the simulated evaluation, we decided to implement the IRL policy iteration, symmetry-exploiting LSPI, EBAC and auto-identification in Simulink for the use with the real platform.

7. Experiments

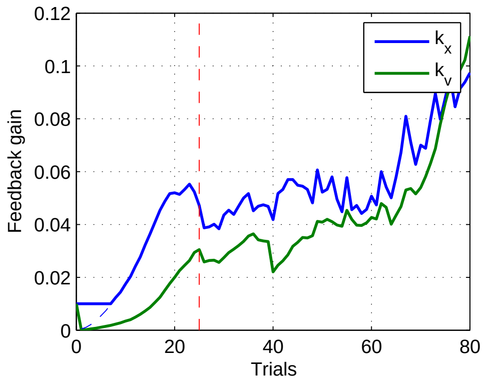
In this chapter we will present the results achieved with the real magnetic platform. The real-time control is implemented in Simulink models using Real-Time Windows Target and xPC Target libraries, which ensure precise timing, meeting the required sampling periods.

7.1. Online IRL value iteration

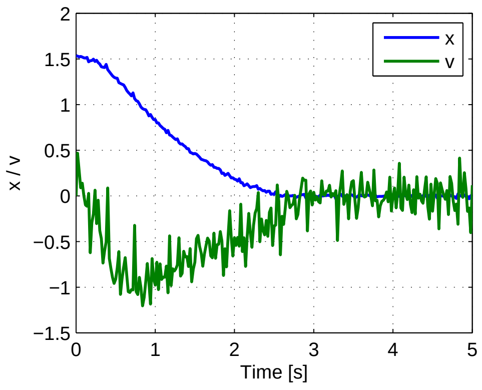
Based on the simulation results, we rejected the policy iteration and decided to implement the value iteration for the real platform. The initial learning experiments failed, because the value iteration increases the feedback gains very slowly at the beginning (see Figure 6.3c), which turned out to be a problem with the real platform. Even if we initialize the value iteration with a more aggressive initial control policy, the algorithm first decreases the feedback gains before converging to the solution. In the simulations, even the low gains are sufficient to start moving the ball. On the real platform, however, there is some minimal force magnitude needed to make the ball move, because the surface of the touch foil bends below the ball and slightly prevents the ball from rolling. The small initial control forces therefore do not move the ball at all, which results in collecting data sets that are useless for current policy evaluation and all of the policy updates are consecutively rejected.

Finally, we decided to initialize the algorithm with a very low initial gain that stabilizes the ball and makes it roll at least slowly to generate some data for policy evaluation and update. Furthermore, we use the low initial gain for position as a minimal allowed gain in the updated policy. The element k_x of the feedback gain vector $\boldsymbol{K} = [k_x, k_v]$ is set at least to the initial gain value after every policy update, which prevents the algorithm from decreasing the gain too much and losing possibility to collect data for improving the control policy. The constraint on gain k_x is put into effect only during the first few trials before the algorithm starts setting a higher value itself. The gain k_v is not constrained in any way and can be set arbitrarily during the policy update.

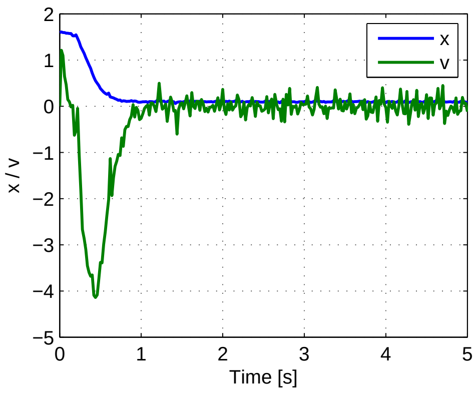
The algorithm is able to reach a good stabilizing solution with the constrained gain k_x. The solution is different from the theoretical optimal solution computed for an ideal double integrator, but the difference is expectable because the real system does not behave exactly the same as the double integrator. However, if not stopped when the gains reach relatively steady levels, the algorithm continues updating the policy, which results in deterioration of the control performance and can eventually lead to the instability of the closed loop system. Figure 7.1 illustrates a typical behavior of



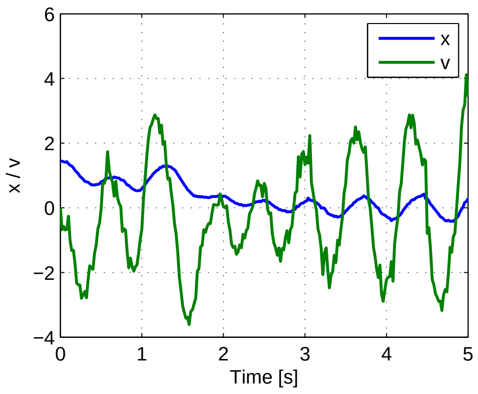
(a) feedback gains during learning



(b) performance with initial policy



(c) performance with final policy



(d) deteriorated performance

Figure 7.1. The IRL value iteration learning on the PC platform with the real magnetic manipulator. Figure (a) shows the development of feedback gains during learning. The dashed blue line at the beginning shows the values that were constrained by the initial gain. The red dashed line marks the trial where the learning would be stopped if we did not want to show the further behavior of the algorithm. Figure (b) shows the performance under the initial stabilizing policy for comparison with the performance under the policy at the moment of stopping the learning in Figure (c). Figure (d) presents the unstable behavior of the closed-loop system during the 80th trial.

the learning algorithm including the phase of policy deterioration. The deterioration is caused by the fact that when the control policy is already performing well, the trial data that is used for policy update loses its richness, which leads to bad parameter estimation. In addition, there is also the correlation of measurement noise and control signal due to the feedback control. The moment when the policy updates should be stopped can be detected as the moment when the policy update attempts to decrease both k_x and k_v at once. The update stopping moment (marked with red dashed line in Figure 7.1a), divides the feedback gain transients into the initial smooth interval of convergence and the subsequent divergence due to the low richness of data. It is interesting that the initial control policy does not produce a rich data either, but the first feedback gain update together with the constraint on k_x results in an oscillatory closed-loop response, which drives the convergence further on.

The execution of a single learning trial was implemented as a Simulink model both on the PC and Speedgoat platform. The whole learning process is controlled by a MATLAB code that launches the Simulink model, downloads and processes the experimental data and waits for manual reset of the system in a loop. The value iteration performs well on the PC, while on the Speedgoat platform with the same settings it gets stuck immediately after the first policy update due to the noisy position measurement. The stronger noise makes the estimate of the $\boldsymbol{P}$ matrix different than negative definite, so the policy updates are rejected. Setting a more aggressive initial policy (and therefore also a higher limit for the gain k_x) does help to overcome the obstacle, but then the difference between the initial and final policy is not so remarkable as on the PC platform, so we do not consider the algorithm useful then.

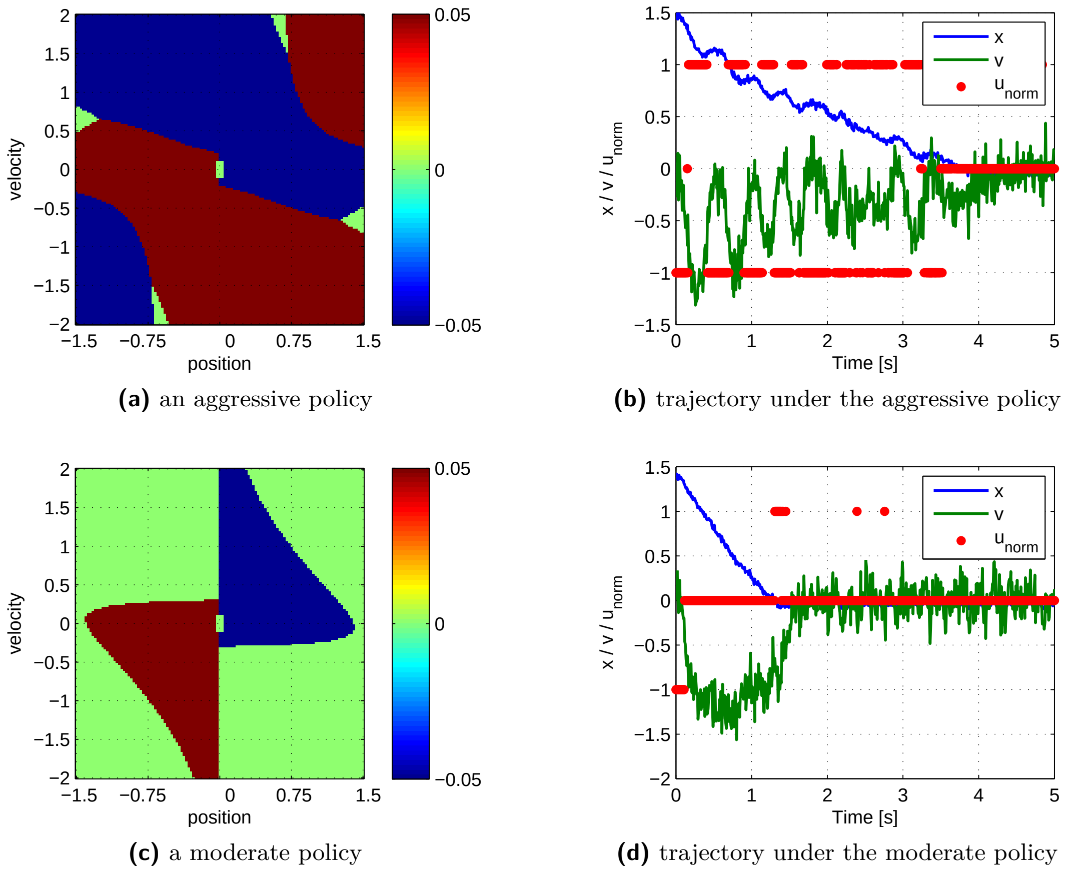
7.2. Online LSPI

The symmetry-exploiting LSPI implementation was fully converted into a Simulink model. Thanks to the ϵ-greedy exploration, the algorithm needs manual reset of the ball position only after a trial that results in stabilization of the ball in the middle of the platform or in the cases when the ball remains out of the area in which it is controllable by the coils.

In the initial experiments, we attempted to let the learning algorithm even hit the border of the magnetic platform with the ball. We had an idea that the controller may learn to use the borders of the platform for controlled bouncing of the ball at situations when it is already not possible to stabilize the ball without hitting the border. Although we managed to simulate such behavior, the bounces destroyed the learning on the real platform. Unlike in the simulation, where the bounces were deterministic, they are quite different one from each other on the real platform. The variability of bounces violates the Markov property of the system, which is one of the basic assumptions for the reinforcement learning.

Our first idea how to prevent the ball from hitting the border was to introduce an extra penalization for the area that is closer to the border of the platform than a certain threshold. The learning algorithm should behave more carefully then in order to avoid the extra penalization. However, this approach appeared not to be suitable because it introduces a discontinuity in the Q-function, which is approximated by smooth basis functions. The learning focused on avoiding the contact with the boundary, but it neglected the stabilization objective. In addition, the ball could still hit the border as a result of an exploratory action.

Instead of the extra penalization, it is possible to end a trial immediately when the ball's position gets out of defined bounds. When we detect a bound violation, we do not generate the next reward value neither update the policy nor the Q-function estimate, but we reset the state and start a new trial, which turned out to be a suitable solution of the problem.



(a) an aggressive policy

(b) trajectory under the aggressive policy

(c) a moderate policy

(d) trajectory under the moderate policy

Figure 7.2. Examples of LSPI policies learned on Speedgoat machine with the real magnetic platform. The Figure (a) shows an aggressive control policy, which results in a slow and oscillatory stabilization. The policy in Figure (b) is more energy-efficient, which paradoxically leads to a faster state transient.

Beside the trial truncation, we also increased the amplitude of the discrete control actions from $U_m = 0.03$ which was used in simulations to $U_m = 0.05$. The larger amplitudes of the control actions simplify the learning process for the algorithm, because their influence is more remarkable in the noisy data.

The real-model experiments run with the PC have shown that even though the algorithm sometimes converged to a control policy that would be able to control a model in simulation, it did not stabilize the ball in reality or produced an oscillatory behavior. On the Speedgoat platform, where we were able to reach the sampling frequency 120 Hz compared to the 50 Hz on the PC, the algorithm is able to learn to stabilize the ball in approximately two out of three runs. The failures can be caused by a combination of bad exploration with measurement noise, because the measurement noise affects not only the learning, but also the feedback linearization. As a result, the platform exerts a different than commanded force, which can confuse the learning. The resulting control policies can vary in behavior quite a lot. Two examples of stabilizing policies with sample controlled trajectories are presented in

Figure 7.2. The policy in Figure 7.2a is quite aggressive, leaving only a small part of the phase plane with zero-control. The trajectory in Figure 7.2b apparently oscillates around the switching line between the positive and negative control action area in the phase plane. Although the response is slow and oscillatory, the aggressive policy can stabilize the ball from any position above the row of coils if the initial velocity of the ball is zero. With a non-zero initial velocity, the ball may get into an area of the phase plane where the policy is not stabilizing (the upper right and lower left corner of the displayed region). The policy in Figure 7.2c is much more efficient than the aggressive one. We can also see that it can stabilize the ball faster. On the other hand, it does not stabilize the ball from the borders of the platform even with zero initial velocity.

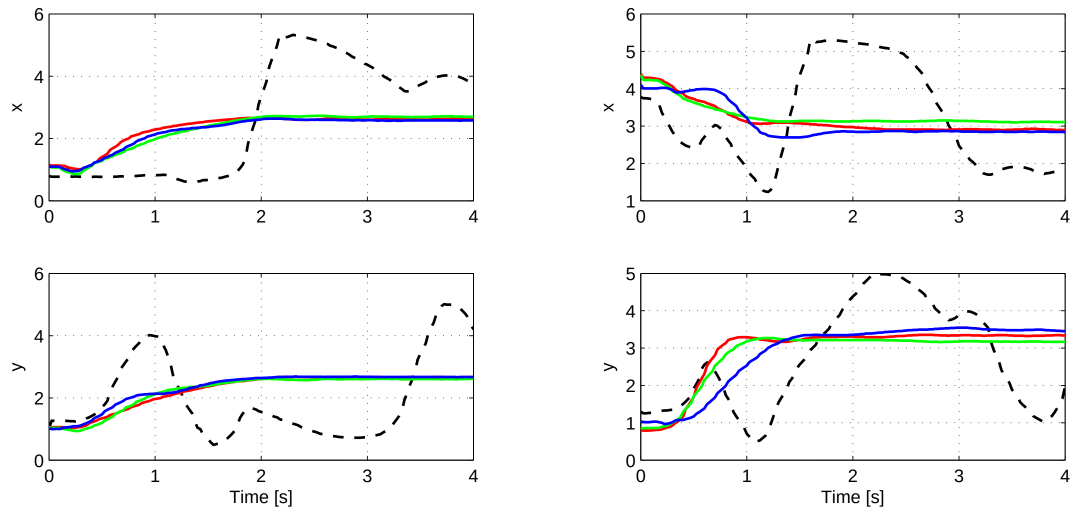
7.3. Online EBAC

The Matlab EBAC implementation was transformed into a Simulink model. The original EBAC framework uses structures that are not supported for Simulink code generation, so we did not attempt to preserve the universality of the original framework and exploited the knowledge of the controlled system to simplify the source code.

When we started the experiments on the PC platform, we did not achieve satisfactory results. The algorithm converged to a policy that expelled the ball to the borders of a platform. We thought that it was necessary to adjust the learning parameters to different values than the ones used in simulations. However, the parameter tuning did not bring any improvements. In addition, the manual parameter tuning with the real platform is quite time-demanding, so we abandoned the EBAC after a bunch of attempts and focused on improving the other methods. When revising the results of the experiments, we discovered a serious bug in the Simulink implementation, which prevented the EBAC algorithm from correct execution. Therefore we do not present the experimental results for the EBAC here. From time reasons, the repetition of the experiments remains for future work.

7.4. Auto-identification

We have used balls of two different sizes for the verification of the auto-identification approach. First, it was the 30 mm ball, which was used for the experiments with reinforcement learning, and second, it was a 20 mm ball. The bigger ball weights 110 g, while the smaller ball only 32.7 g, which makes it more difficult to control. In addition, we have to use the delayed position measurement from the camera, because the small ball cannot be detected by the touch foil. The visual position measurement delay does not affect the identification of the acceleration profile, which is done offline, but the subsequent control of the ball. We were also limited by the maximal achievable sampling rate of the visual measurement. The smaller ball is more sensitive to the exerted magnetic force, so it can experience accelerations of magnitude around



(a) easy evaluation transient

(b) difficult evaluation transient

Figure 7.3. Evaluation responses of the ball position to a step change of reference position. The transients in the x- and y-direction are shown separately for better readability. The red line marks the performance of the PIDf controller with the 30 mm ball using the analytical model of the force profile. The green line and the blue line mark the performance with the 30 mm ball, resp. 20 mm ball using the corresponding auto-identified acceleration profile. Finally, the black dashed line shows the behavior of the 20 mm ball being controlled with the analytical force profile for 30 mm.

10 m.s^{-2}. The estimated magnitudes of acceleration are lower than the real ones with a too low sampling frequency. We were aware that we would get biased estimates for the small ball with the maximal sampling rate 50 Hz, so we energized the attracting coil with activation factor only 0.5 during the identification experiment and then normalized the resulting acceleration profile by the activation factor, i.e. multiplied it by the factor of 2.

The evaluation experiments succeeded only on the PC platform, because Speedgoat does not allow to use the camera and its noisy touch foil position measurement is not suitable for acceleration estimation. In addition, we would not be able to measure the position of the smaller ball using the touch foil. The experiments consist of two-dimensional control of the position of a ball. The evaluation experiments use the same PIDf controller and differ in the size of the controlled ball and usage of analytical or auto-identified force/acceleration profile. We compare the original control of the 30 mm ball which uses the manually identified analytical force profile with the suggested auto-identification scheme applied to both 30 mm and 20 mm ball. To emphasize the effect of the auto-identification, we present also transient measured with the 20 mm ball using the force profile for the 30 mm ball. Figure 7.3 illustrates the four variants of two different transients of the ball's position that were chosen for evaluation. First, it is the response to the step of reference position from platform coordinates $\boldsymbol{x} = [1,1]^T$ to the center of the platform $\boldsymbol{x} = [2.5, 2.5]^T$. Second, it is the response to the step of reference position from platform coordinates $\boldsymbol{x} = [4,1]^T$

to position $\boldsymbol{x} = [3, 3]^T$. The first transient in Figure 7.3a is easier for the controller, because the ball moves along the diagonal of the platform, so the off-diagonal coils are distributed symmetrically with respect to the position of the ball and the imprecisions of their contributions that would drive the ball off the diagonal cancel out. The other transient shown in Figure 7.3b already reveals the imprecision of the identified acceleration profile for the 20 mm ball. Especially the x-coordinate transient differs significantly from the ones that were measured for the 30 mm ball. On the other hand, there is a noticeable contribution of the auto-identification, because the switch from the 30 mm to the 20 mm ball causes instability without the use of the auto-identified model.

7.5. Conclusions

We have converted the MATLAB implementation of the selected algorithms to Simulink models that allow execution with the real magnetic manipulator, including the interconnection of the magnetic manipulator with Speedgoat machine. We ran the experiments and succeeded to deploy the IRL value iteration and auto-identification on the PC and the LSPI on Speedgoat platform.

The IRL value iteration on Speedgoat machine was limited by the increased position measurement noise caused by the power supply of the magnetic platform, but it worked on the PC with the same settings of the algorithm. It is possible that after removing the problem by using the laboratory power supply, it will be possible to deploy the IRL value iteration on Speedgoat platform as well.

The LSPI was deployed only on Speedgoat platform, because it turned out that the learning required a sampling frequency 120 Hz, which was not achievable on the PC. The algorithm has proven a certain robustness, because it was able to learn even under the presence of the increased measurement noise.

The auto-identification with the real platform succeeded in controlling both balls of different sizes using the same settings for the controller using the feedback linearization parametrized with identified acceleration profiles. However, the control performance with the 30 mm and 20 mm ball apparently differed from each other. The difference is caused by imprecise acceleration profile identification due to the limited position measurement sampling frequency for the real system. The subsequent feedback control with the identified acceleration profiles also suffered from the delay of the visual position measurement. We could not use the touch foil measurement with the smaller ball, because it does not generate sufficient pressure on the surface of the platform to be detected.

8. Conclusion

The main tasks of the thesis were to design a system for visual measurement of position of manipulated objects, explore the possibilities of learning and adaptive control methods and deploy them on the laboratory model of the magnetic platform connected to a PC or Speedgoat target machine.

The designed computer vision system brought the possibility to measure positions of one or more balls simultaneously, but it reaches the selected target sampling frequency for visual feedback control (40 Hz) only when detecting one or two balls. The main drawback of the visual position measurement is its significant delay (approximately 27 ms), which could not be eliminated, because its major part is caused by the propagation of the video signal from a framegrabber card to Simulink environment. A possible solution is to use an external device for fast implementation of the ball detection and propagate only the detected positions to Simulink. For example, a camera with a built-in FPGA board like some of the devices manufactured by Optomotive company could be used.

The tested reinforcement learning methods performed well in noise-free simulations, although we had to abandon the initial idea of learning a policy along a single trajectory and switch to the trial learning due to the requirement of persistent excitation. In the subsequent experiments, we managed to deploy two of the selected methods on the real magnetic platform, namely the IRL value iteration on PC and the LSPI on Speedgoat platform.

The IRL value iteration on Speedgoat was unfortunately affected by the increased noise of position measurement caused by Magman's power supply. The strong noise caused that the estimated solution $\boldsymbol{P}$ of the Riccati equation was not negative definite as required, so the algorithm rejected all policy updates.

The experiments showed that the success of learning can depend not only on the quality of measurements of states provided to the reinforcement learner, but also on the sampling frequency, because the LSPI run on Speedgoat machine with sampling frequency 120 Hz was able to converge to a solution even with the more noisy position measurement compared to the PC with better position measurement sampled at 50 Hz, where the LSPI failed. With sampling frequency decreased to 50 Hz, the LSPI did not converge on Speedgoat platform either. Therefore we think that it would be useful to put more effort in improving the position measurement in the future to get a solution that will be at once less noisy and fast, because some of the learning methods may require high sampling rates in order to work effectively.

The reinforcement learning methods that use random exploration during the learning process also face a problem of missing guarantees for performance of the learned policy. The problem is caused by the limited amount of exploration, which is attenuated during learning or stopped after some learning period elapses, because it deteriorates the performance of the learned policy. Although the exploration attenuation is set to provide the algorithms enough time for converging to a good solution, we still only rely on the fact that the algorithms converge before the exploration vanishes.

Similarly to the reinforcement learning, the suggested auto-identification would also profit from a better position measurement. Especially for the identification of the acceleration profile for the 20 mm ball, it would be useful to have a higher sampling frequency available than 50 Hz from the visual position measurement. The subsequent control with the auto-identified acceleration profiles suffered also from the position measurement delay.

Although the Speedgoat rapid prototyping platform has its limitations, we find it quite handy. First, it is because of the variety of interfaces that allow connecting different peripherals easily. Second, Speedgoat provides high performance and exact synchronization during the execution of target applications thanks to the real-time kernel. Finally, the target application development with Speedgoat is definitely faster and easier than writing one's own target applications e.g. in C, even with the relatively lengthy code generation, compilation and target initialization of the Simulink models, which seemed clumsy at first.

8.1. Future work

First of all, the future work includes repeating the experiments with Speedgoat platform together with the low-noise resistive foil measurement improved by using a stabilized laboratory power supply instead of Magman's power supply. It is possible that the control algorithms will perform better.

Second, the experiments with the corrected Simulink implementation of EBAC should be performed to evaluate its behavior with the real magnetic platform.

Third,within the scope of this thesis, we managed to accomplish the task of one-dimensional stabilization of a single ball above a row of coils using two different reinforcement learning methods, so the logical next steps are the command following (still in one dimension) and the subsequent extension to the two-dimensional control of one ball, afterwards extended to the parallel control of more balls. Every extension will increase the dimensionality and therefore also the difficulty of the learning problem. The parallel manipulation will be challenging also because of the possible interaction of the manipulated objects. Beside increasing the dimensionality of the controlled system, it would be also interesting to use the reinforcement learning for direct coil control without the feedback linearization layer.

Appendix A.

Basler acA2000-340kc with BitFlow NEON-CLB

The Camera Link camera and the framegrabber have a specific way of settings. Both of them come with a specialized software from their manufacturer and can be configured independently. However, their configurations have to be consistent in order to work correctly together.

A.1. Camera settings

The camera is configured through Basler Pylon Viewer software, which allows adjusting various parameters from the exposure settings to the area of interest (AOI) to the parameters of serial communication with the framegrabber card. All the camera settings that are available in Pylon Viewer form a configuration set. It is possible to store four different configuration sets in the non-volatile internal memory of the camera. One of the configuration sets is used as the default startup set, which specifies the default parameter values that are set every time the camera reboots. The default configuration set is the only way how to adjust the camera when being used with Speedgoat platform, because Speedgoat cannot run Pylon Viewer.

A.2. Camera configuration file

The framegrabber needs information about the connected camera including the number of active pixels, bit depth and possibly the hardware triggering options. The information is contained in the camera configuration file and has to fit the current camera settings adjusted through Pylon Viewer, otherwise the acquired image is broken apart. The camera configuration files can be browsed and edited using CamEd utility from BitFlow SDK. Beside the camera information, the configuration files also allow turning on or off the hardware Bayer decoding on the framegrabber, as well as the Power over Camera Link (PoCL) option, which can power the camera from the framegrabber card without the need for an external power adapter.

Appendix B.

Speedgoat Performance Real-Time Platform

B.1. Target machine settings

The target machine settings are accessible through a graphical interface of the xPC Explorer, which is launched by the command `‘xpcexplr’` in Matlab on the host PC. The settings can be changed only when the target machine is disconnected and they are persisted on the target machine by updating its kernel. The kernel is transferred on a USB stick in a following procedure:

1. Save the settings in the xPC Explorer.
2. Connect the USB stick to the host PC (suppose that it connects e.g. as drive ‘E:’) and generate updated boot files for the target machine by the command `‘speedgoatmachineboot(‘E:’)’`.
3. Plug the USB stick into the target machine and boot it. The target machine will automatically load the updated kernel with settings.
4. After a prompt, remove the USB stick from the target machine and reboot it with the new settings.

B.2. Camera Link support

Although we could not use the Camera Link camera for experiments due to the missing Bayer decoding support, we have an interesting remark that applies also to monochromatic cameras. The Simulink block for the video acquisition using BitFlow Neon framegrabber is parametrized with a camera configuration file, but it does not allow to adjust the camera settings like the exposure time, bit depth, image sensor gain etc., which are usually set using a software provided by the manufacturer of the camera. As the software cannot be installed on Speedgoat, one needs a PC with a compatible framegrabber card to pre-configure the camera in advance, persist the settings in its registers and then connect it to Speedgoat. The Basler cameras have a default startup configuration set that can be used for this purpose.

B.3. Data logging & file transfers

The measured signals can be logged to Speedgoat's local HDD using the xPC File Scope block from Simulink library. The files can be afterwards downloaded to the host PC for further processing. The easiest way how to transfer the files between the target machine and host PC is to use the `xpctarget.ftp` object as

```
ftp = xpctarget.ftp;
ftp.get('fileName.dat');
```

to store the data file from the target machine in the current Matlab directory on the host PC. The data is stored in binary xPC Target file format, which can be loaded into a structure in Matlab workspace using a specialized utility

```
dataStructure = readxpcfile('fileName.dat');
```

Although it is quite common to save the captured video-sequences when experimenting with a camera, the xPC Target does not provide a reasonable tool to save a video signal. Although the video can be transmitted to the host PC over UDP, this option works only for low resolution and frame rate of the video. When we used a USB webcam that produced a 640×480 px RGB signal at 30 frames per second, the resulting frame rate on the host PC after UDP transmission was 3 frames per second. We therefore attempted to save the video to Speedgoat's local HDD through the xPC File Scope, but even this solution is not suitable, because every pixel intensity value is processed as a single signal by the File Scope, resulting in tens of thousands of signals. Compilation and initialization of a target application for saving a 204×204 px video through the File Scope takes approximately 17 minutes. For a higher resolution, the compilation was terminated with an error. In conclusion, it is not possible to save a high resolution video using the xPC Target.

Appendix C.

Contents of the attached CD

A CD with Matlab source codes and other materials is attached to the thesis. The content of the CD is organized in following directories:

- *Thesis/*: contains electronic version of the thesis text
- *MATLAB/control*: the implementation of the discussed control algorithms
- *MATLAB/magSimulink*: the Simulink library with basic blocks for Magman
- *MATLAB/vision*: the implementation of the computer vision

Bibliography

[1] P. Abbeel, A. Coates, M. Quigley, and A.Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *In Advances in Neural Information Processing Systems 19*, page 2007. MIT Press, 2007.

[2] A. Al-tamimi, M. Abu-Khalaf, and F. Lewis. *Machine Learning*, chapter Heuristic Dynamic Programming Nonlinear Controller. InTech, 2009.

[3] J.G. Allen, R.Y.D. Xu, and J.S. Jin. Object tracking using camshift algorithm and multiple quantized feature spaces. Technical report, School of Information Technologies, University of Sydney, 2006.

[4] D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 3rd edition, 2007.

[5] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1st edition, 1996.

[6] R.R. Bitmead. Persistence of excitation conditions and the convergence of adaptive schemes. *Information Theory, IEEE Transactions on*, 30(2):183–191, Mar 1984.

[7] K.F. Böhringer and H. Choset. *Distributed Manipulation.* Springer US, 2000.

[8] K.F. Böhringer, B.R. Donald, and N.C. MacDonald. Single-crystal silicon actuator arrays for micro manipulation tasks. In *Micro Electro Mechanical Systems, 1996, MEMS '96, Proceedings. An Investigation of Micro Structures, Sensors, Actuators, Machines and Systems. IEEE, The Ninth Annual International Workshop on*, pages 7–12, Feb 1996.

[9] K.F. Böhringer, B.R. Donald, and N.C. MacDonald. What programmable vector fields can (and cannot) do: Force field algorithms for mems and vibratory plate parts feeders. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 822–930, 1996.

[10] G.R. Bradski. Computer vision face tracking for use in a perceptual user interface. *Intel Technology Journal*, 2, 1998.

[11] L. Busoniu, D. Ernst, B. De Schutter, and R. Babuška. Online least-squares policy iteration for reinforcement learning control. In *American Control Conference (ACC), 2010*, pages 486–491, June 2010.

[12] L. Busoniu, D. Ernst, B. De Schutter, and R. Babuška. Approximate reinforcement learning: An overview. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on*, pages 1–8. IEEE, 2011.

[13] H. Byun and S.W. Lee. Applications of support vector machines for pattern recognition: A survey. In *Pattern recognition with support vector machines*, pages 213–236. Springer, 2002.

[14] P. Corke. *Robotics, Vision and Control*, chapter Planar Homography. Springer Berlin Heidelberg, 2011.

[15] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[16] V. Duindam, A. Macchelli, S. Stramigioli, and H. Bruyninckx. *Modeling and Control of Complex Physical Systems*. Springer, 2009.

[17] G.A. Dumont and M. Huzmezan. Concepts, methods and techniques in adaptive control. In *American Control Conference, 2002. Proceedings of the 2002*, volume 2, pages 1137–1150 vol.2, 2002.

[18] M. Fairbank and E. Alonso. The divergence of reinforcement learning algorithms with value-iteration and function approximation. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8, June 2012.

[19] H. Funaya and K. Ikeda. A statistical analysis of soft-margin support vector machines for non-separable problems. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–7, June 2012.

[20] P.Y. Glorennec. Reinforcement learning: an overview. In *European Sym. on Intelligent Techniques*, 2000.

[21] Speedgoat GmbH. Support – Target Machine Setup Guides. `http://www.speedgoat.ch/TargetMachines_Landing.aspx`, March 2014.

[22] G.J. Gordon. Reinforcement learning with function approximation converges to a region. In *Advances in Neural Information Processing Systems*, pages 1040–1046. The MIT Press, 2001.

[23] I. Grondman, M. Vaandrager, L. Busoniu, R. Babuška, and E. Schuitema. Efficient model learning methods for actor-critic control. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 42(3):591–602, June 2012.

[24] H. Hjalmarsson, M. Gevers, S. Gunnarsson, and O. Lequin. Iterative feedback tuning: theory and applications. *Control Systems, IEEE*, 18(4):26–41, Aug 1998.

[25] P. Holoborodko. Noise robust differentiators for second derivative estimation. `http://www.holoborodko.com/pavel/numerical-methods/numerical-derivative/smooth-low-noise-differentiators`, March 2014.

[26] Z. Hurák and J. Zemánek. Feedback linearization approach to distributed feedback manipulation. In *American Control Conference (ACC), 2012*, pages 991–996, June 2012.

[27] A. Hyo-Sung, Ch. Yang-Quan, and K.L. Moore. Iterative learning control: Brief survey and categorization. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(6):1099–1121, Nov 2007.

[28] I.D. Landau, R. Lozano, and M. M'Saad. *Adaptive control.* Springer, 1998.

[29] I. Leichter, M. Lindenbaum, and E. Rivlin. Mean shift tracking with multiple reference color histograms. *Computer Vision and Image Understanding*, 114(3):400 – 408, 2010.

[30] F.L. Lewis and K.G. Vamvoudakis. Optimal adaptive control for unknown systems using output feedback by reinforcement learning methods. In *Control and Automation (ICCA), 2010 8th IEEE International Conference on*, pages 2138–2145, June 2010.

[31] F.L. Lewis, D. Vrabie, and K.G. Vamvoudakis. Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. *Control Systems, IEEE*, 32(6):76–105, Dec 2012.

[32] L. Matignon, G.J. Laurent, and N. Le Fort-Piat. Design of semi-decentralized control laws for distributed-air-jet micromanipulators by reinforcement learning. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 3277–3283, Oct 2009.

[33] M. Piccardi. Background subtraction techniques: a review. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 4, pages 3099–3104 vol.4, 2004.

[34] W.B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* John Wiley and Sons, 2007.

[35] D.K. Prasad. Survey of the problem of object detection in real images. *International Journal of Image Processing*, 6, 2012.

[36] A. Salhi and A.Y. Jammoussi. Object tracking system using camshift, meanshift and kalman filter. *World Academy of Science, Engineering and Technology*, 64:674–679, 2012.

[37] E. Schuitema. *Reinforcement learning on autonomous humanoid robots.* PhD thesis, TU Delft, November 2012.

[38] O. Sprangers, G.A.D. Lopes, and R. Babuška. Reinforcement learning for port-hamiltonian systems. 2013. Submitted, available at `http://arxiv.org/abs/1212.5524`.

[39] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 28. MIT press, 1998.

[40] K.G. Vamvoudakis and F. Lewis. Online actor-critic algorithm to solve the continuous-time infinite horizon optimal control problem. *Automatica*, 46:878–888, 2010.

[41] H. van Seijen and R. S. Sutton. True online td(λ). In *ICML '14: Proceedings of the 31th International Conference on Machine Learning*, 2014. To appear.

[42] W.J.R. Velthuis. *Learning feed-forward control*. PhD thesis, University of Twente, February 2000.

[43] D. Vrabie and F. Lewis. Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems. *Neural Networks*, 22:237–246, 2009.

[44] D. Vrabie, O. Pastravanu, M. Abu-Khalaf, and F. Lewis. Adaptive optimal control for continuous-time linear systems based on policy iteration. *Automatica*, 45:477–484, 2009.

[45] F.-Y. Wang, H. Zhang, and D. Liu. Adaptive dynamic programming: An introduction. *Computational Intelligence Magazine, IEEE*, 4(2):39–47, May 2009.

[46] S. Wang, J. Braaksma, and R. Babuška. Reinforcement learning control for biped robot walking on uneven surfaces. In *Proceedings of the 2006 International Joint Conference on Neural Networks*, pages 4173–4178, 2006.

[47] Y. Wang, F. Gao, and F.J. Doyle. Survey on iterative learning control, repetitive control and run-to-run control. *Journal of Process Control*, 19:1589–1600, 2009.

[48] A. Yilmaz, O. Javed, and S. Mubarak. Object tracking: A survey. *ACM Computing Surveys*, 38(4), 2006.

[49] J. Zemánek and Z. Hurák. Experimental modular platform for distributed planar manipulation by shaping magnetic field. *IEEE/ASME Transactions on Mechatronics*, 2014. to be submitted.