Dynamic Bat-Control of a Redundant Ball Playing Robot

Dennis Schüthe

Kumulative Dissertation
zur Erlangung des Grades eines
Doktors der Ingenieurwissenschaften – Dr.-Ing. –

Vorgelegt im Fachbereich 3 (Mathematik und Informatik)
Universität Bremen

25.08.2016

Datum des Promotionskolloquiums: 02. Februar 2017

Gutachter

Prof. Dr. Udo Frese (Universität Bremen)
Prof. Dr. Axel Gräser (Universität Bremen)

Abstract

This thesis shows a control algorithm for coping with a ball batting task for an entertainment robot.

The robot is a three jointed robot with a redundant degree of freedom and its name is "Doggy". Doggy because of its dog-like costume. Design, mechanics and electronics were developed by us. DC-motors control the tooth belt driven joints, resulting in elasticities between the motor and link. Redundancy and elasticity have to be taken into account by our developed controller and are demanding control tasks.

In this thesis we show the structure of the ball playing robot and how this structure can be described as a model. We distinguish two models: One model that includes a flexible bearing, the other does not.

Both models are calibrated using the toolkit Sparse Least Squares on Manifolds (SLOM) – i. e. the parameters for the model are determined. Both calibrated models are compared to measurements of the real system.

The model with the flexible bearing is used to implement a state estimator – based on a Kalman filter – on a microcontroller. This ensures real time estimation of the robot states. The estimated states are also compared with the measurements and are assessed. The estimated states represent the measurements well.

In the core of this work we develop a Task Level Optimal Controller (TLOC), a model-predictive optimal controller based on the principles of a Linear Quadratic Regulator (LQR). We aim to play a ball back to an opponent precisely. We show how this task of playing a ball at a desired time with a desired velocity at a desired position can be embedded into the LQR principle. We use cost functions for the task description. In simulations, we show the functionality of the control concept, which consists of a linear part (on a microcontroller) and a nonlinear part (PC software). The linear part uses feedback gains which are calculated by the nonlinear part.

The concept of the ball batting controller with precalculated feedback gains is evaluated on the robot. This shows successful batting motions.

The entertainment aspect has been tested on the Open Campus Day at the University of Bremen and is summarized here shortly. Likewise, a jointly developed audience interaction by recognition of distinctive sounds is summarized herein.

In this thesis we answer the question, if it is possible to define a rebound task for our robot within a controller and show the necessary steps for this.

Zusammenfassung

Diese Arbeit zeigt einen Regelalgorithmus zur Bewältigung einer Ballspielaufgabe für einen Unterhaltungsroboter.

Der Roboter besteht aus drei Drehgelenken mit einem redundanten Freiheitsgrad und hört auf den Namen „Doggy“ – Doggy wegen seines hundeähnlichen Kostüms. Design, Mechanik und Elektronik wurden von uns entwickelt. DC-Motoren steuern die Zahnriemen getriebenen Gelenke und dies führt zu Elastizitäten zwischen Motor und Gelenk. Redundanz und Elastizität müssen von dem entwickelten Regler berücksichtigt werden, was eine herausfordernde Aufgabe ist.

Wir zeigen in dieser Arbeit den Aufbau des ballspielenden Roboters und wie dieser als Modell beschrieben werden kann. Dabei unterscheiden wir zwei Modelle: Eines berücksichtigt das flexible Kugellager, das andere nicht.

Beide Varianten werden mit Hilfe des Tools Sparse Least Squares on Manifolds (SLoM) kalibriert – d. h. die Parameter für das Modell werden bestimmt. Die Kalibrierungen werden mit Messungen des realen Systems verglichen.

Aus dem Modell mit dem flexiblen Lager wird ein Zustandsschätzer – basierend auf einem Kalman Filter – auf einem Mikrocontroller implementiert. Dieser sorgt für Echtzeitschätzung der Roboterzustände. Die geschätzten Zustände werden ebenfalls mit den Messungen verglichen und bewertet. Zustände und Messungen stimmen dabei sehr gut überein.

Im Kernpunkt dieser Arbeit entwickeln wir einen Task Level Optimal Controller (TLOC), ein modellprädiktiven optimaler Regler, der auf den Prinzipien eines Linear Quadratic Regulator (LQR) beruht. Wir verfolgen das Ziel, einen Ball gezielt zum Mitspieler zurück zu spielen. Wir zeigen wie diese Aufgabe, einen Ball zu einer bestimmten Zeit mit bestimmter Geschwindigkeit in einer bestimmten Position zu spielen, in das LQR-Prinzip eingebettet werden kann. Zur Aufgabenbeschreibung nutzen wir Kostenfunktionen. In Simulationen zeigen wir die Funktionalität des Reglerkonzepts, welches aus einem linearen Teil (auf Mikrocontrollerebene) und einem nichtlinearen Teil (PC Software) besteht. Der lineare Teil nutzt dafür Rückführgrößen, die vom nichtlinearen Teil berechnet werden.

Der TLOC Algorithmus wird mit vorberechneten Rückführgrößen auf dem Roboter evaluiert. Dies zeigt ein gelungenes Ausführen von Schlagbewegungen.

Der Unterhaltungsaspekt von Doggy wurde auf dem Open Campus Tag der Universität Bremen getestet und wird hier kurz präsentiert. Ebenso stellen wir eine Publikumsinteraktion vor, bei welcher Doggy auf markante Geräusche reagiert.

Wir beantworten in dieser Arbeit die Frage, ob eine Ballrückschlag-Aufgabe für unseren Roboter innerhalb eines Reglers definiert werden kann und zeigen die erforderlichen Schritte hierfür.

Acknowledgment

This work has been supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

I want to thank SyDe and especially Prof. Frese for giving me the opportunity to work on such an interesting PhD project. I also want to thank Prof. Gräser and Dr. Kassahun for being part of my project committee and the regularly discussions within it. The discussions I had with Prof. Pannek were helpful and I thank him for it. Building the robot and electronics could only be done with the help of Alexis Maldonado, Jens Hilljegerdes, and Christoph Budelmann – they did their bit to build the big.

Special thanks to Felix Wenk for the cooperation in the calibration. We had great discussions on the calibration and additionally for the Kalman filter implementation.

There are some people to name that had great effort of getting me this thesis done. I thank all my friends, especially Isabella and Silvia for her help during the hard days of my PhD and for proofreading. Moreover, there is to name Svenja and Bettina that read through the thesis for corrections, this was very helpful. My girlfriend Verena just for being there – it is nice to have someone who bolsters me.

Finally, I want to thank my family for their great support over the years. Without this support, this work would never have been possible. You, my siblings and parents, have a huge share on this work.

Contents

List of Figures

List of Tables

Acronyms

ARE . . . . . algebraic Riccati equation

CAD . . . . . computer aided design

CCF . . . . . cross correlation function

COG . . . . . center of gravity

DOF . . . . . degree of freedom

EKF . . . . . Extended Kalman Filter

EOF . . . . . end-effector

GPIO . . . . . General Purpose Input Output

HRI . . . . . human-robot interaction

IC . . . . . itegrated circuit

IMU . . . . . Inertial Measurement Unit

KF . . . . . Kalman Filter

LQG . . . . . Linear Quadratic Gaussian

LQR . . . . . Linear Quadratic Regulator

MHT . . . . . Multiple-Hypothesis-Tracker

MPC . . . . . Model predictive control

pwm . . . . . pulse width modulation

RRT . . . . . Rapidly-exploring Random Trees

SLoM . . . . . Sparse Least Squares on Manifolds

TLOC . . . . . Task Level Optimal Control

UKF . . . . . Unscented Kalman Filter

WCS . . . . . world coordinate system

List of Symbols

A global notation within this thesis is to mark vectors $\boldsymbol{v}$ as bold symbols and in general the vectors are column vectors. Matrices $\boldsymbol{M}$ are written in uppercase bold. The transpose $\boldsymbol{v}^T$ and the inverse $\boldsymbol{M}^{-1}$ is denoted by a superscript mark. A diagonal matrix can be written as $\operatorname{diag}(\boldsymbol{v}) = \begin{pmatrix} v_1 & 0 & 0 \\ 0 & v_2 & 0 \\ 0 & 0 & v_3 \end{pmatrix}$. Some symbols might be used for a different purpose, which is explained in the text. The main usage of a symbol is given is the table below.

Notation	Symbol	Description
$\boldsymbol{\theta}$	rad	Motor link position
$\boldsymbol{\theta}_\mathrm{m}$	rad	Motor position
$\dot{\boldsymbol{\theta}}$	$\mathrm{rad\,s^{-1}}$	Motor angular link velocity
$\dot{\boldsymbol{\theta}}_\mathrm{m}$	$\mathrm{rad\,s^{-1}}$	Motor angular velocity
$\ddot{\boldsymbol{\theta}}$	$\mathrm{rad\,s^{-1}}$	Motor angular link acceleration
$\ddot{\boldsymbol{\theta}}_\mathrm{m}$	$\mathrm{rad\,s^{-1}}$	Motor angular acceleration
$\boldsymbol{\vartheta}$		Parameter vector
$\boldsymbol{\mu}_\mathrm{fm}$	N m	Motor friction coefficient
$\boldsymbol{\Sigma}$		Covariance matrix
$\boldsymbol{\tau}_\mathrm{b}$	N m	Coupling torque of the spring between motor and link
$\boldsymbol{\tau}_\mathrm{c}$	N m	Coupling torque of the spring between motor and link
$\boldsymbol{\tau}_\mathrm{cfb}$	N m	Coupling torque of the spring between motor and link including the flexible bearing
$\boldsymbol{\tau}_\mathrm{fl}$	N m	Link friction
$\boldsymbol{\tau}_\mathrm{fm}$	N m	Motor friction
$\boldsymbol{\tau}_g(\boldsymbol{q})$	N m	Gravity Force
$\boldsymbol{\tau}_\mathrm{m}$	N m	Motor torque link side
$\boldsymbol{\tau}_\mathrm{ms}$	N m	Motor torque motor side
$\boldsymbol{\tau}_\mathrm{s}$	N m	Coupling torque of the spring between motor and link
$\boldsymbol{\omega}$	$\mathrm{rad\,s^{-1}}$	Angular velocity
$\boldsymbol{A}$		State transition matrix
$\boldsymbol{B}$		Command matrix
$\boldsymbol{b}_\mathrm{m}$	N m s	Motor inertia on the link side
$\boldsymbol{b}_\mathrm{ms}$	N m s	Motor inertia
$\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}})$	N m	Coriolis Force

Notation	Symbol	Description
$\boldsymbol{D}_\mathrm{s}$	$\mathrm{N\,m\,s}$	Spring damping matrix constant between motor and link
$\boldsymbol{d}_\mathrm{s}$	$\mathrm{N\,m\,s}$	Spring damping vector constant between motor and link
$\boldsymbol{d}_\mathrm{sfb}$	$\mathrm{N\,m\,s}$	Spring damping vector constant between motor and link, and in the flexible bearing
$f_\mathrm{dyn}(\boldsymbol{x}, \boldsymbol{u})$		Dynamics function
$f_\mathrm{kin}(\boldsymbol{x})$		Kinematics function
I_m	A	Motor current
I_max	A	Maximum current
$\boldsymbol{K}$		Feedback gain
k_mi	$\mathrm{N\,m\,s}$	Mutual induction constant translates motor speed to torque
k_pv	s	Translates motor velocity to a mutual induction pulse width modulation (pwm) signal
k_pwm	$\mathrm{N\,m\,V^{-1}}$	Constant translates a voltage to a torque
$\boldsymbol{K}_\mathrm{s}$	$\mathrm{N\,m}$	Spring matrix constant between motor and link
$\boldsymbol{k}_\mathrm{s}$	$\mathrm{N\,m}$	Spring vector constant between motor and link
$\boldsymbol{k}_\mathrm{sfb}$	$\mathrm{N\,m}$	Spring vector constant between motor and link, and in the flexible bearing
k_τ	$\mathrm{N\,m\,A^{-1}}$	Translates motor current to torque
$\boldsymbol{M}(\boldsymbol{q})$	$\mathrm{kg\,m^2}$	Link Inertia Matrix
$\boldsymbol{M}_\mathrm{fb}(\boldsymbol{q}_\mathbf{fb})$	$\mathrm{kg\,m^2}$	Link Inertia Matrix with flexible bearing extension
p_0		PWM zero
$\boldsymbol{p}_\mathrm{d}$	m	Desired position vector
P	W	Power
$\boldsymbol{P}$		Accumulated weight
$\boldsymbol{p}$		PWM signal for each motor, $p_i \in [-1, 1]$
$\boldsymbol{Q}$		State weighting/penalizing matrix
$\boldsymbol{q}$	rad	Link Position
$\dot{\boldsymbol{q}}$	$\mathrm{rad\,s^{-1}}$	Link Velocity
$\ddot{\boldsymbol{q}}$	$\mathrm{rad\,s^{-1}}$	Link Acceleration
$\boldsymbol{R}$		Command weighting/penalizing matrix
r_g		Gear ratio between input gear (motor) and output gear (link)
R_m	Ω	Motor resistance
${}_{from}^{to}\boldsymbol{R}$		Rotation matrix from coordinate system *from* to coordinate system *to*
$\boldsymbol{S}$		State-command weighting/penalizing matrix
${}_{from}^{to}\boldsymbol{t}$		Translation vector from coordinate system *from* to coordinate system *to*
${}_{from}^{to}\boldsymbol{T}$		Combines translation and rotation from coordinate system *from* to coordinate system *to*
T_s	s	Sample Time
$\boldsymbol{u}$		Command vector

Notation	Symbol	Description
U_m	V	Motor voltage
U_max	V	Maximum voltage
$\boldsymbol{v}_\mathrm{d}$	$\mathrm{m\,s^{-1}}$	Desired velocity vector
w		World Coordinate System
x		Coordinate axis – colored red
$\boldsymbol{x}$		State vector
y		Coordinate axis – colored green
z		Coordinate axis – colored blue
$\boldsymbol{Z}$		Stacked measurements

1 Introduction

Imagine, someone throws a ball in your direction and your task is to hit the ball back with your head. Could you do that? For us as humans this task can be handled with a big variance of precision, depending on our age – a child of five would probably not know how to do it; our experiences – a football player would probably do best; and much more. We see that it could be quite difficult for us to fulfill this task, but in most cases we manage to do it. And we manage it because we think of the task in a global way. We consciously not decompose the task into several steps, like predicting where the ball will be in the future, and if the ball is close to me how should I move my head, or should I better use my legs to step to the side? Would it be better to move with high velocity or should I just stand still and hope the ball drops back? We do not think about all this, we are just reacting to the task we were given. For a robot this is a lot more difficult and mostly divided into subtasks – tracking the ball, making a prediction, planning a trajectory of the motion to hit the ball, and so on. That is why tasks like Ping Pong playing are used to demonstrate these algorithms – because the tasks are very demanding. This is why we ask the question

Is it possible to define a rebound task for our robot within a controller?

This question includes that the controller decides autonomously what the best way is to realize that task in an optimal sense, i. e. not decompose the task into trajectory planning followed by trajectory control. The controller should be fully responsible for the reaction of a thrown ball. To be more precise, we want the robot to fulfill the task of being at a desired time t_d at a desired position $\boldsymbol{p}_\mathrm{d}$ having a desired velocity $\boldsymbol{v}_\mathrm{d}$ which is needed to hit the ball back. The implementation should be done for an entertainment robot called “Doggy” (see Figure 1.1) that has been built and designed by us.

Figure 1.1.: Ball playing entertainment robot Doggy.

In this thesis we show how such a demanding task can be put into an optimal controller that also utilizes the robot's redundancy and exploits it to fulfill the task. The controller itself uses a model and controls the robot's state. The model was identified by a calibration which is also part of this thesis. The contributions made are given below, followed by an outline of this thesis, and a state of the art in entertainment and ball playing robotics.

1.1. Contributions

We contributed a Task Level Optimal Control formulation of a ball batting task on an entertainment robot, showed its behavior in simulations and on the robot. The proposed controller should be able to handle similar tasks. Another contribution was made by the calibration procedure where we only use a minimalist sensor setup (encoders and gyroscopes, Section 4). We used the identified model to implement a state estimator in form of a Kalman filter that runs in real time on a microcontroller.

1.2. Outline

An overview about the parts described within this thesis and their relations is shown in Figure 1.2. The colors denote chapters and the boxes are modules implemented within

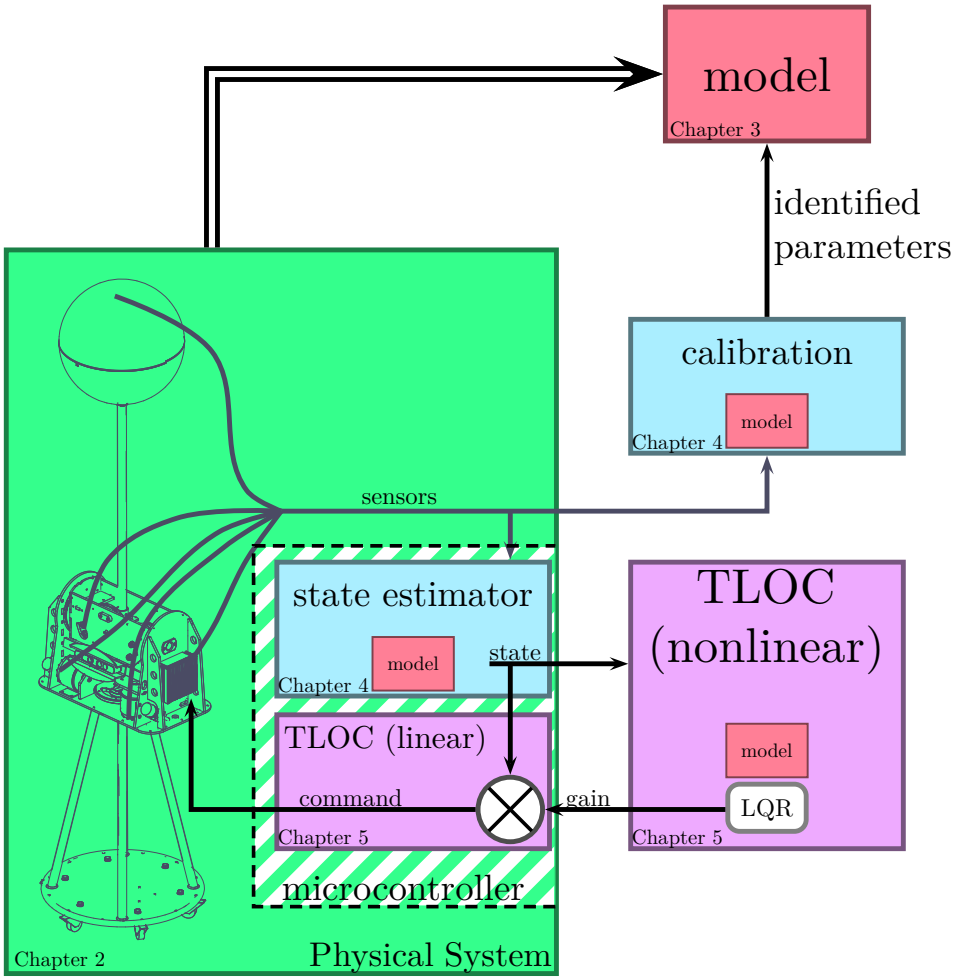


Figure 1.2.: Thesis Overview with descriptions of the Chapters.

this thesis. We will start with a description of the robotic system in Chapter 2, which explains the mechanical system, the sensors, the camera system, and the electronic system. This system is formalized into a model in Chapter 3. The model is essential for the state estimator and the Task Level Optimal Control. Additionally, it is used in the calibration to identify the model parameters for our robot (Chapter 4). Due to the similar behavior, this chapter presents the implemented state estimator. The Task Level Optimal Control algorithm is based on a Linear Quadratic Regulator (LQR) and uses the identified model for model predictive control (Chapter 5). Herein the implementation and task description is given and tested on simulations and the robot itself. Having the entertainment aspect in mind we give an overview about an exhibition the robot was presented at and about its ability for human-robot interaction (Chapter 6). Finally, we conclude and give some ideas for future work (Chapter 7).

Our entry point is the state of the art of entertainment and ball playing robotics. Related work on the specific topics – Chapters 4 and 5 – is part of the chapters.

1.3. State of the Art

In this section the ball playing and entertainment aspect is related to other work, i. e. what other entertainment robots are out there and what kind of other ball playing robots are there? How do they implement the ball playing task? Here, we only give an overview of the classification of the overall concept.

First of all there is to name the robot "Piggy", which is a predecessor version of Doggy. Piggy has only two servo motors as joints, which make the ball tracking system fixed to one view [Laue et al. 2013]. In this paper the overall concept for that robot is explained. The whole concept slightly changed for the new version of Doggy and the related software has been adapted. The new overview and concept of Doggy has been published in [Schüthe 2015]. In robotics the entertainment aspect grew only slowly within the last years. Entertainment systems to name are mostly within the artificial pets or toy area. E. g. the *Sony AIBO* [Pransky 2001] – which broke the record of robots sold in the shortest time – and the *RoboSapiens* [Tilden 2004]. The former is a home entertainment robot with artificial intelligence. It simulates a dog's behavior of walking, playing, emotions, and learning. While the latter is a small humanoid robot, which is used also for playing soccer [Behnke et al. 2006]. Another entertainment robot developed by *Sony* is the *QRIO* [Ishida 2004]. This humanoid robot has played golf and conducted an orchestra [Geppert 2004].

The human-robot interaction (HRI) and entertainment aspect is combined in the work of [Schraft et al. 2001]. Three robots – the "*Inciting*", the "*Instructive*", and the "*Twiddling*" – are put into a museum environment to fulfill different aspects. The first one welcomes new visitors to the museum and memorizes them for a given time period. The second acts as a tour guide for the visitors and guides them through the exhibition and gives explanations on the exhibits. The last one is designed like a child with three moods. It looks for a ball and is happy as long as it sees it. If the ball is out of focus it changes its mood to grumpy and angry depending on the time since the last ball has been seen. Moreover, the robots are able to interact with each other.

In connection with human ball playing interaction, there is to name the *Segway-Soccer* [Argall et al. 2006]. A team of humans standing on Segways is playing against a robotic Segway team, this eliminates most disadvantages of the robot compared to its human opponents. Another ball playing system is the *KiRo* [Weigel et al. 2003], which plays table soccer against humans. Here, the same mechanism as for the Segways is used – a system which brings human and robot to the same level of playing.

Numerous articles can be found for table tennis ball games using different approaches for the control. In the last few years the machine learning aspect got a lot of attention [Matsushima et al. 2005; Mülling et al. 2013; Silva et al. 2015]. Other table tennis systems using different control approaches are [Andersson 1989; Serra et al. 2016; Yamakawa et al. 1989; Yang et al. 2010].

The first table tennis robot mentioned can be found in [Andersson 1986]. The reason for the great interest is probably the variety of technical challenges that can be solved. This experimental setup is simple and well known, so it is understood by most groups

of people. However, this is also a risk because playing table tennis is just a simple task for humans. For our experimental setup we were also looking for a simple setup which gets the attention of children, not technical, and technical enthusiasts.

Except for table tennis, which has the widest range, there are different robot ball players out there. A volleyball playing robot attached to the ceiling and not portable[Nakai et al. 1998]. In the baseball major league a humanoid pitched the first ball [Lofaro et al. 2012]. A vision-system for HRI with ball playing as the interaction has been proposed in [Yamaguchi et al. 2003]. The machine of development should kick the ball back to the player. In [Hu et al. 2010] a robot throws a basketball into the basket. In the papers' attached video the robot wears a costume in form of a seal. Batting a fast ball to a desired point on a high speed motion is explained in [Senoo et al. 2006] by using a small racket on a robot arm – baseball like. Catching flying balls is investigated in [Bäuml et al. 2010, 2011; Deguchi et al. 2008; Riley et al. 2002] as HRI.

The most recent example of entertainment has been shown at the Eurovision Song Contest 2016. Three KUKA robots performed a dance battle against humans during the act "Man vs. Machine". KUKA got an amazing feedback for the performance and told that people are watching it over and over again [KUKA 2016]. This shows the emotions that can be created by entertainment robots. Moreover, there are Rock Bands made of robots, like the bands "Compressorhead" [*Compressorhead* 2016] and "Z Machines" [ZMachines 2015]. Both are bands that play famous rock songs on real instruments and also give concerts.

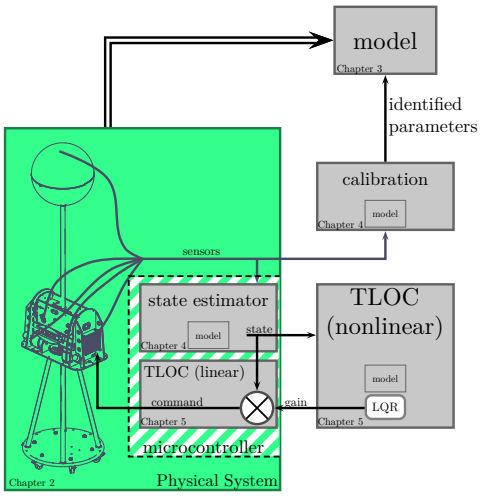
[Erdmann 2013] gives an overall view into the topic sport robotics based on biomechanics. He also presents different types of sport robots and shows their applications for sport communities.

All these showing the challenges of object recognition, motion planning, control, getting in touch by playing a game with the audience, and a lot more. The following two examples are the most related ones, focusing on the entertainment part and making the robot part of a group that plays ball games.

Kober et al. described the challenge of physical interaction and contact between human and robot in theme park environments. It is mentioned that ball games – here in form of catching and throwing – are a form of physical engagement while maintaining a save distance between human and robot. They use a humanoid that catches balls from participants and throws them back to them. For some participants they tested also the juggling between human and robot with three balls. The robot is able to interact with the person by reacting on lost balls with gestures and following the opponent with its head [Kober et al. 2012a]. A video shows the similarity to our entertainment robot [Kober et al. 2012b].

The most famous entertainment robot to name is *RoboKeeper* [KG 2016]. This system is a robotic goalkeeper which is available for hire commercially. A lot populism is taken to that goalkeeper and lots of videos exist where it plays against football professionals, like Lionel Messi. The goalkeeper has also been adapted to different variants, i. e. Hockey and Handball. The system is kept simple. A vision tracking system mounted above the goal tracks the balls with 90 images per second. The balls color must distinguish from

its background. Via an image processing software the impact point can be computed to move the keeper to that position. The challenge is the speed, because a kicked ball can accelerate to more than 100 km/h leaving the tracking and moving only 0.3 s to react. To make it fair for the audience, the keeper can be adjusted to one of seven difficulty levels. This robot with its well described task of interacting with an audience and entertaining them gets closest to our imagination of an entertainment robot. And we can see a lot of similarities here, although the *RoboKeeper* is held even simpler. But it also shows that an entertainment system does not have to be very complex to get a lot of attention.



2 The Robotic System

The goal of this chapter is to understand the overall design of our robot. The idea of the robot is to have a minimalistic system consisting of a sphere to hit a ball, a 2-degree of freedom (DOF) workspace with an additional third axis to pan the cameras and to rotate to the audience. This system can entertain people on events, like on the OPEN CAMPUS day of University of Bremen in 2015[1].

First, let us take a look into the robotic system to understand the details that are relevant to the controller for reference. We start with the mechanical structure and the integration of sensors (Section 2.1). The controller deals with elastic joints which are explained in Section 2.2. In the following section we take a look at the motors used to drive the robot and report on some engineering problems solved in this thesis. A basic discussion on how to estimate the link position using Inertial Measurement Units (IMUs) is given in Section 2.4. A brief explanation on the camera system for ball detection and prediction, which is not part of this thesis, is found in Section 2.5. The chapter concludes with an explanation of the electronic system, where the basic structure of the control and the reason for using a distributed system of microcontroller and computer are given. Moreover, it illustrates the work done on electronics and software.

2.1. Mechanical Structure and Sensor Integration

The robot consists of four bodies which are connected via three revolute joints in a kinematic chain. By definition, revolute joints turn around their z-Axis [Craig 2005; Waldron et al. 2008]. When talking about joint positions and velocities in this thesis, I

[1] A video of it can be viewed on `http://www.informatik.uni-bremen.de/agebv2/downloads/videos/doggyOpenCampusDay.mp4`. Note that we used a simple position controller for the movements which is not part of this thesis.

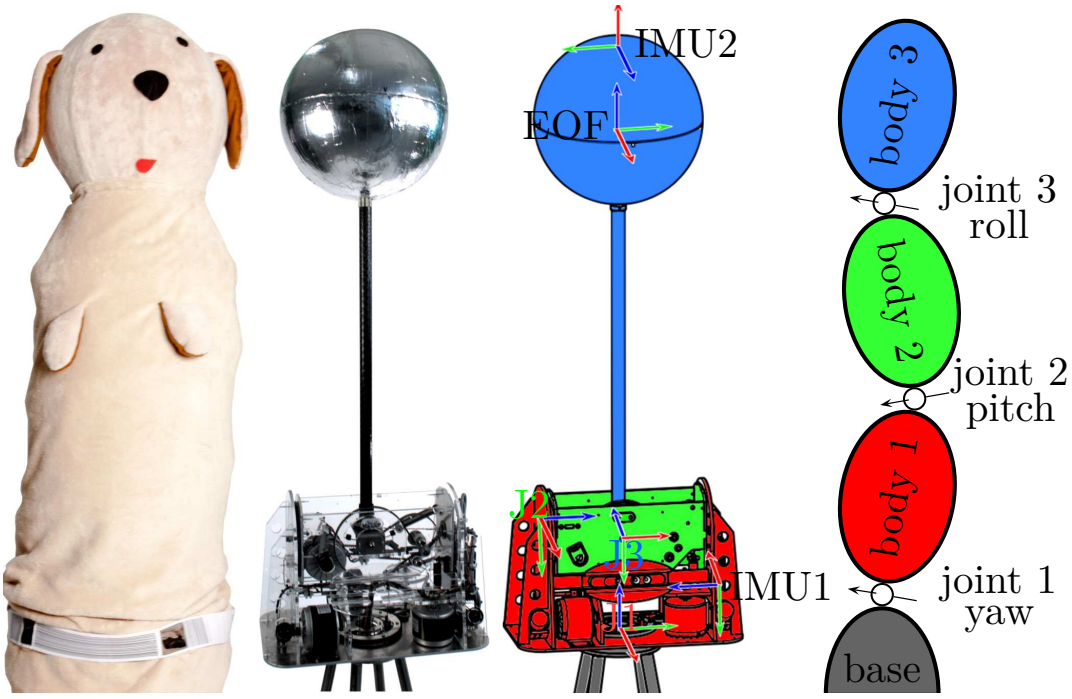


Figure 2.1.: Doggy's CAD drawing marked with colors for each body. Coordinate systems are represented in color notation red x-axis, green y-axis, and blue z-axis (left). An abstract definition of the joints and bodies is on the right.

refer to the angular positions and angular velocities. For now and throughout the thesis all axes are color encoded with the definition: red for x-axis, green for y-axis, and blue for z-axis.

We start with the third body – Doggy's head, which is our end-effector (EOF) used as a racket. The head consists of a 40 cm Styrofoam sphere and includes a mount for an IMU. The IMU in the head is one of two IMUs used to estimate the joint's link positions $\boldsymbol{q}$ and velocities $\dot{\boldsymbol{q}}$. The IMUs coordinate system I2, as well as all other coordinate systems are shown in Figure 2.1 together with an abstract representation of all joints and bodies. A carbon rod is attached to the head – to basically define the radius of the workspace – and to the third joint J3. The third joint rolls the EOF. Additionally, a pitch joint J2 is coupled to the EOF by the second body and third joint. Joints two and three together span with the EOF a partial hollow sphere, the robots workspace, where all the points on this sphere are possible hit positions. The second body includes the motor and gear for the third joint. Moreover, joints 2 and 3 have a spring attached between the tooth wheel and a fixed part of the holding body. This counteracts the gravity force for both joints.

The first body is the biggest and holds: (1.) A left plate with a bearing to mount joint two, and the microcontroller circuit board where IMU I1 is put on. (2.) The right plate with another bearing and the active part of the second joint, i. e. a tooth wheel to turn the second body. (3.) Motors and gears for first and second joint. (4.) Servo motors

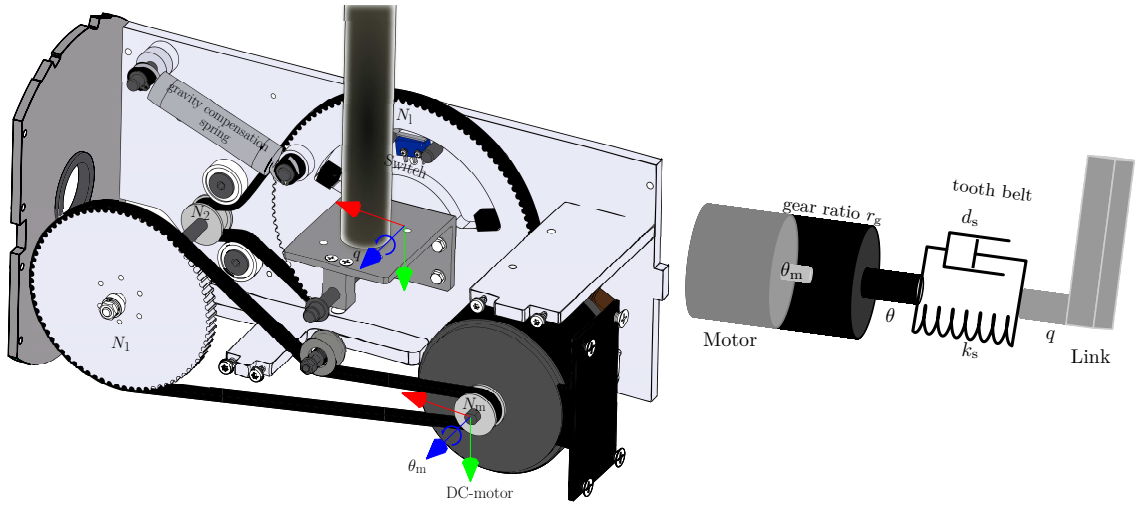


Figure 2.2.: The drive mechanism between motor and link is shown besides the abstract joint model connecting the motor and link via a spring damper system.

for Doggy's tail actuation. And finally, (5.) a stereo camera system. The first body is turned by the yaw joint J1. The predefined workspace can be turned by this joint to enlarge the workspace, as the limitations for joint three and two are different. The yaw joint is a redundant degree of freedom, which is both a challenge and an opportunity for optimal control. Also, this joint connects the first body with the base. The base is fixed in the world and defines the base coordinate system. It holds the power supply and a computer.

To move the robot, DC motors drive the links by tooth belts. This demands a deeper look inside the dynamics, because we get elasticity into the system. Elasticity is hard to handle in a controller and could be reduced by better mechanical design. However, we take this as a challenge and ask: "*Is it possible to deal nicely with elasticities in an optimal controller?*" The result is this tooth belt driven system to test the controller on it.

The setting of an elastic joint is described in the next section, including the gears between motor and link.

2.2. Elastic Joints

Let us exemplify this on the third axis as representative for the other axes. Figure 2.2 shows the setting of the second body holding the driving part of joint three. The z-Axis points towards the rotational axis of the joint. In this case the motor axis (marked with a blue arrow) points towards the same direction, which leads to the same turning direction. This is the convention used in [Craig 2005] where joint axis and frame z-Axis coincide. When looking on the axis a positive turn means left, a negative turn right in a mathematical sense. The total gear ratio is the relation of output to input tooth number

– here in two stages:

$$r_\mathrm{g} = \frac{75}{14}\frac{110}{14} = 42.09 \tag{2.1}$$

This ratio has to be taken into account when motor and link values are compared. An encoder on the motor measures the position (Section 2.3). The position of the motors $\boldsymbol{\theta}_\mathrm{m}$ and their velocities $\dot{\boldsymbol{\theta}}_\mathrm{m}$ can be transformed to the link side by $\boldsymbol{\theta} = r_\mathrm{g}\boldsymbol{\theta}_\mathrm{m}$ and $\dot{\boldsymbol{\theta}} = r_\mathrm{g}\dot{\boldsymbol{\theta}}_\mathrm{m}$, respectively. This is the position before the tooth belt's elasticity (Figure 2.2).

Let us now define the coupling between motor and link. Two scenarios we can easily imagine. First, there is no connection between motor and link. The force that the motor transfers to the link would be zero and vice versa. Secondly, motor and link are directly connected on the same axis. Then the force of the motor will directly be transferred to the link and vice versa. Moreover, the link and motor positions would be identical, i. e. $\boldsymbol{\theta} \equiv \boldsymbol{q}$. So taking a tooth belt as coupling must be something in between. And the tighter the tooth belt is tensioned the more it acts as a direct coupling. This coupling can be approximated by a spring damper system with spring constant k_s and damping constant d_s (Figure 2.2) as described in [De Luca et al. 2008]. If the stiffness is set to infinity, link and motor positions are equal. A stiffness of zero means no connection between link and motor. The elasticity is a problem because it makes control more indirect and it requires to estimate link positions $\boldsymbol{q}$ and velocities $\dot{\boldsymbol{q}}$ in addition to motor positions $\boldsymbol{\theta}$ and velocities $\dot{\boldsymbol{\theta}}$. The coupling torque of this spring can be expressed as

$$\boldsymbol{\tau}_\mathrm{c} = \boldsymbol{K}_\mathrm{s}\left(\boldsymbol{q} - \boldsymbol{\theta}\right) + \boldsymbol{D}_\mathrm{s}\left(\dot{\boldsymbol{q}} - \dot{\boldsymbol{\theta}}\right) \quad . \tag{2.2}$$

The damping $\boldsymbol{D}_\mathrm{s} = \mathrm{diag}\left(\boldsymbol{d}_\mathrm{s}\right)$ and spring $\boldsymbol{K}_\mathrm{s} = \mathrm{diag}\left(\boldsymbol{k}_\mathrm{s}\right)$ constant matrices are of diagonal form holding values for each joint and thus can be presented as vectors $\boldsymbol{k}_\mathrm{s} = \left(\begin{smallmatrix} k_{\mathrm{s,J1}} & k_{\mathrm{s,J2}} & k_{\mathrm{s,J3}} \end{smallmatrix}\right)^T$ and $\boldsymbol{d}_\mathrm{s} = \left(\begin{smallmatrix} d_{\mathrm{s,J1}} & d_{\mathrm{s,J2}} & d_{\mathrm{s,J3}} \end{smallmatrix}\right)^T$. If no external force acts to this system the torque is zero, which is the equilibrium of the spring.

2.3. Motors

The robot's motion is created by brushed DC motors driving the joints through tooth belts. In the first robot version "Piggy" servo motors were used, which had the disadvantage of insufficient torque. Also their teeth were broken after a while due to the impact of the ball. In addition, the speed was very limited.

We decided to use scooter motors, as they are provided in the low cost segment and they come with a sufficient torque. Moreover, a brushed DC motor is easier to handle as a brushless motor, where a phase shifted signal must be provided. In the low cost segment no sensors are provided on the motor. To control the behavior of the motors we need sensors which tell us the position or velocity. It turned out, that the motors could easily be equipped with sensors. Therefore, the motor's back side was drilled to get access to the shaft. To hold an encoder, we 3D printed an adapter which can be fit to the motor's back. For the coupling between motor shaft and encoder we drilled a

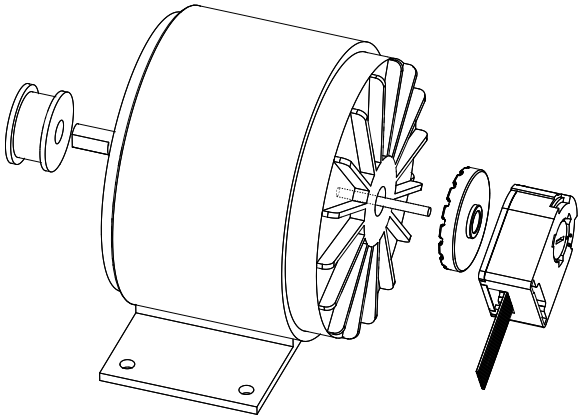


Figure 2.3.: Motor disassembly view. Tooth wheel in front. Axis extension shaft, 3D printed plate – connects motor and encoder – and the encoder on the back.

hole into the motor shaft and inserted a smaller shaft of 3 mm that fits into the encoder. This structure is shown in Figure 2.3.

To drive the motor a pulse width modulation (pwm) signal is used, where the voltage is modulated within $\pm U_{\max} = \pm 32\,\mathrm{V}$. A single chip H-Bridge driver switches the power according to the microcontroller pwm[2].

2.3.1. Current Limitation

The motor current depends on the applied pwm, the motor resistance R_m, and the motor velocity $\dot{\theta}_\mathrm{m}$.

$$I_\mathrm{m} = \frac{U_{\max}\left(p - k_\mathrm{pv}\dot{\theta}_\mathrm{m}\right)}{R_\mathrm{m}} \tag{2.3}$$

The constant value k_pv translates the velocity into a pwm signal, which is the mutual induction of the motor [Vukosavic 2013]. If in a free running motor the pwm is hold constantly, the velocity will also be constant and produces a mutual induction pwm which equals the constant pwm ($p = k_\mathrm{pv}\dot{\theta}_\mathrm{m}$) such that the motor current gets zero. We call this mutual induction pwm that produces zero current the pwm zero p_0.

$$p_0 = k_\mathrm{pv} \cdot \dot{\theta}_\mathrm{m} = 0.159\,59\dot{\theta}_\mathrm{m} \tag{2.4}$$

To obtain the parameter k_pv we measure the motor speed in free running mode for given pwm ($p \in [-1, 1]$), where a negative pwm turns the motor clockwise and a positive value turns the motor counter clockwise (view on the motor axis, see also [Vukosavic 2013]).

[2]The data sheet suggests that the IC automatically limits motor current by applying reverse voltage. However, this does not work when changing the direction of motion. Also, during the motion change the motor becomes a generator for a short time and feeds back voltage, i. e. voltage rise. So it has to be handled manually.

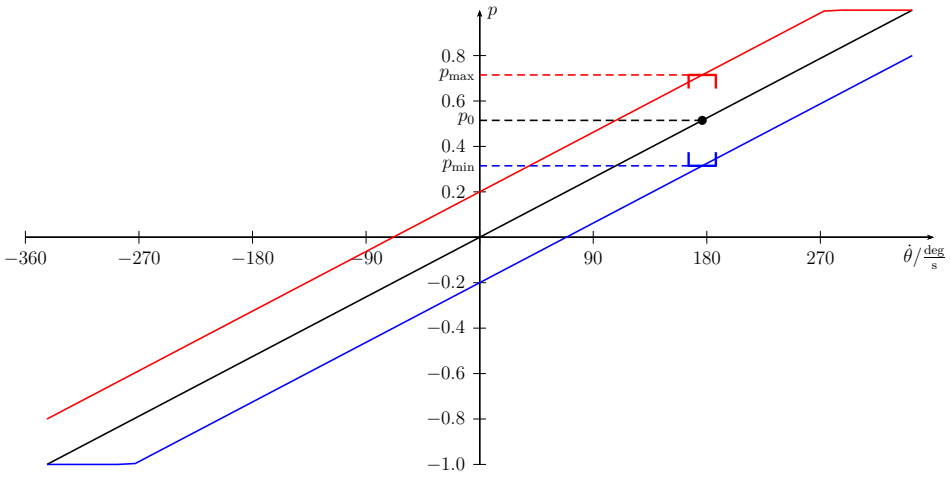


Figure 2.4.: Current limitation using the motor velocity $\dot{\boldsymbol{\theta}}$ to compute the zero pwm p_0 (black line). The current is limited around p_0 in a range of $p_0+0.2$ (red line) and $p_0-0.2$ (blue line), this is equivalent to a current limitation of $\pm 5\,\text{A}$.

To limit the current we only allow a change of $\Delta p_{\max}$ around p_0. We set the maximum allowed current to $I_{\max} = 5\,\text{A}$ to have a safety margin to the maximum the motor driver chip accepts, i. e. 7 A.

$$\Delta p_{max} = \pm\frac{I_{\max} R_{\text{m}}}{U_{\max}} = \pm\frac{5\,\text{A}\cdot 1.28\,\Omega}{32\,\text{V}} = 0.2 \tag{2.5}$$

We can now apply a limited pwm p to the motor computed from the pwm p_{set} we set such that

$$\tilde{p} = \begin{cases} p_0 + 0.2 & \text{for } p_{\text{set}} - p_0 > 0.2 \\ p_0 - 0.2 & \text{for } p_{\text{set}} - p_0 < -0.2 \\ p_{\text{set}} & \text{otherwise} \end{cases} \tag{2.6}$$

and afterwards limiting the pwm to the maximum value of 1 (Fig. 2.4) by

$$p = \max\left(1, \min\left(-1, \tilde{p}\right)\right) \quad . \tag{2.7}$$

2.3.2. Motor Braking – Voltage limitation

When braking the motor, i. e. applying a reverse current to the direction of motion, the energy accumulates in supply capacitors and their voltage rises. This leads to an overvoltage fault condition caused by the fact that the motor acts as a generator when decelerating.

To dissipate the overvoltage, we check if the motor is acting as a generator or as a motor. We can compute the power it is producing or consuming in dependency of the pwm and the pwm zero, i. e.

$$P_i = \underbrace{\frac{U_{\max}^2}{R_{\mathrm{m}}}}_{P_{\max}} p_i \left(p_i - p_{0,i}\right) \quad . \tag{2.8}$$

The total power the system consumes is then given by

$$P = \sum_i P_i \quad . \tag{2.9}$$

If this total power gets negative, there is more power produced than consumed and we need to dissipate this to avoid an overvoltage condition. Therefore, if $P < 0$ we switch a power dissipation resistor[3] on in a duty cycle that dissipates $-P$.

2.4. IMU for Link Angle Estimation

By now, we are able to measure the motor positions $\boldsymbol{\theta}$. To measure the link positions $\boldsymbol{q}$ we equipped two IMUs to the robot. One in the head and one on the first body. The first body IMU detects motions of the first axis. The second one detects motions from all axes. The two IMUs are needed to separate the motions for each link.

We make use of the gyroscope on each IMU to detect the angular velocity $\boldsymbol{\omega}$ around the three coordinate axes of the gyroscope. We can combine the information given by gyroscope one ($\boldsymbol{\omega}_1$) and two ($\boldsymbol{\omega}_2$) to obtain the link velocities $\dot{\boldsymbol{q}}$. An integration of the link velocity leads to its position. Each gyroscope has a bias $\boldsymbol{\omega}_0$ which causes a drift in the position (for details see Chapter 4). We avoid this by binding the link position to the motor position via the dynamics function. Using only gyroscopes for link angle estimation is also new to the field of robotics and we can show that this works in our case.

2.5. Camera System

To hit the ball back to a specified position we need to know where the ball is relative to the robot. In our case we want to know the ball's coordinates relative to our world coordinate system, which lies in the center of the robot on the ground. The detection of the ball is only briefly discussed herein as it was a PhD thesis by Oliver Birbach [Birbach 2012].

[3] A solid state relay is used to switch the resistor in dependency of a given duty cycle (pwm) from the microcontroller.

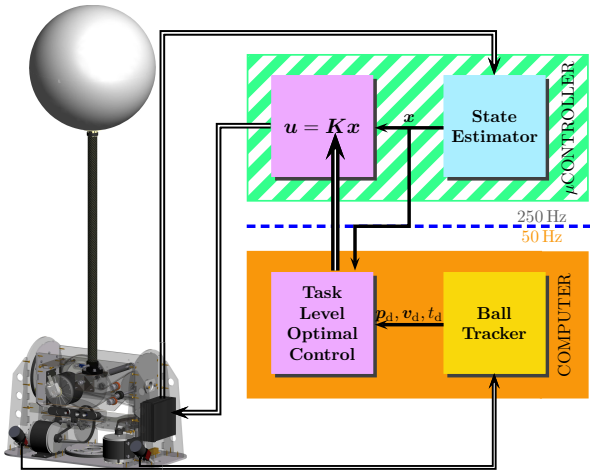


Figure 2.5.: Working principle of the distributed control system. The microcontroller stage runs a fast acting feedback controller in a linear sense. The Computer acts as a nonlinear optimizer of the controller.

In principle: We use a stereo camera based system with a sampling rate of 50 Hz. In each camera image circles of given sizes – around the size of the ball – are searched. The color of the ball does not matter. The detected circles in both images are put into a Multiple-Hypothesis-Tracker (MHT). The MHT combines the circles from both images into a 3D representation. Additionally, the physical model of a flying ball is provided to the MHT and so it can compute how likely the movements of the "circles" are. In addition, the trajectory of the hypotheses are predicted. Hypotheses with a probability below a limit will be removed and only hypotheses with high probability survive. The hypothesis with the highest probability is taken as the ball's trajectory and the prediction is used for computing the intersection point of the ball and the robots workspace.

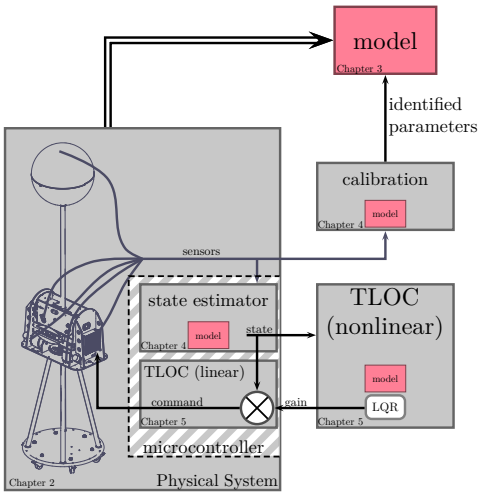
2.6. Electronic System

The control of the motors to move the robot is organized in a two staged system. The first stage runs on a computer, the second stage on a microcontroller. The basic idea is to use the microcontroller for hard real time tasks, i. e. computation of the actual control command, actuating the motors by a pwm signal, reading the motor encoders and the IMU data. The microcontroller acts at a frequency of 250 Hz in a linear manner.

All other processes, i. e. the nonlinear control part of the robot, the computation of control gains, the interaction with audience, and the ball tracking using cameras, run on a computer with much higher computation power, but without a real time operating system. Moreover, we have the discrepancy of the camera sampling rate (50 Hz) and the microcontroller sampling rate of 250 Hz. Our intention was to achieve the elegance of a combination of a fast feedback controller acting linearly on the microcontroller with a nonlinear optimization of the controller running on the computer and updating the

linear feedback controller on a camera sampling basis (Fig. 2.5).

Finally, it should be noted that the microcontroller electronics including motor drivers, IMU, camera synchronization, some power and encoder management was developed in the context of this thesis. This also includes the low level software running on the microcontroller and the computer software that interacts with the microcontroller via USB and Ethernet.



3 Robotic Model

After we have understood the robotic system, we can take the next step and put the information into the kinematics and dynamics, which are essentially for our controller. In the kinematics we put together the information of the coordinates to describe the 3D-position of the EOFs in the world – necessary for our controller in the description of position and velocity differences to the desired ones. The dynamics add the elastic joint model and motor specific characteristics. This is used for calibration and state estimation as well as a model for our controller. In Section 3.3 we discuss an extension of the kinematics and dynamics to deal with flexibilities in the bearing of the first joint.

3.1. Kinematics

Mostly, robots are rigid body systems connected by joints. The rigid body position and orientation in space is called the pose. The kinematics describes the pose and its derivatives of each body [Waldron et al. 2008]. Let us now define the kinematics for our EOF, as this is the information we need for the batting task. Also, it is obvious that in this case the orientation of the head is meaningless as it is a sphere. So, we define the center of the head as position of the EOF given in the world coordinate system (WCS). A detailed description of coordinate transformations is given in Appendix A.1.

The kinematics is the result of the coordinate transforms using the coordinate systems defined in Chapter 2. This gives us a point in the world frame given the link positions $\boldsymbol{q}$, with $\mathrm{c}_i = \cos(q_i)$ and $\mathrm{s}_i = \sin(q_i)$:

$$f_{\mathrm{kin}}(\boldsymbol{x}) = \begin{pmatrix} (\mathrm{c}_1\mathrm{s}_2\mathrm{c}_3 - \mathrm{s}_1\mathrm{s}_3)1010.069\,\mathrm{mm} \\ (\mathrm{s}_1\mathrm{s}_2\mathrm{c}_3 + \mathrm{c}_1\mathrm{s}_3)1010.069\,\mathrm{mm} \\ 1010.069\,\mathrm{mm}\,\mathrm{c}_2\mathrm{c}_3 + 1013.2\,\mathrm{mm} \end{pmatrix} . \tag{3.1}$$

3.2. Dynamics

The dynamics describes the relation between contact force and actuation, and the resultant accelerations and motion trajectories [Featherstone et al. 2008]. The dynamics is essential for control and simulation. They contain the differential equations to describe the acceleration of the motors $\ddot{\boldsymbol{\theta}}$ as a result of the motors torque $\boldsymbol{\tau}_\mathrm{m}$. We have to take into consideration that the torque can be before the gear ratio or afterwards. To keep it simple, we always compute on the link side. The torque transformation from motor to link side is $\boldsymbol{\tau}_\mathrm{m} = r_\mathrm{g}\boldsymbol{\tau}_\mathrm{ms}$. We use the differential equations given in [De Luca et al. 2008]

$$\mathbf{0} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{\tau}_g(\boldsymbol{q}) + \boldsymbol{\tau}_\mathrm{c} + \boldsymbol{\tau}_\mathrm{fl} \tag{3.2}$$

$$\boldsymbol{\tau}_\mathrm{m} = \mathrm{diag}\left(\boldsymbol{b}_\mathrm{m}\right)\ddot{\boldsymbol{\theta}} - \boldsymbol{\tau}_\mathrm{c} + \boldsymbol{\tau}_\mathrm{fm} \ , \tag{3.3}$$

with the coupling between motor and link already defined in Equation (2.2). $\boldsymbol{M}(\boldsymbol{q})$ and $\boldsymbol{b}_\mathrm{m}$ are link and motor inertia respectively, $\boldsymbol{\tau}_\mathrm{fl}$ is the link friction and $\boldsymbol{\tau}_\mathrm{fm}$ the motor friction, $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ are the Coriolis terms, and $\boldsymbol{\tau}_g(\boldsymbol{q})$ are the gravitational terms. Also, the motor inertia has to be transferred to the link side by $\boldsymbol{b}_\mathrm{m} = r_\mathrm{g}^2\boldsymbol{b}_\mathrm{ms}$.

The gravitational terms are obsolete in our case, thanks to the springs acting against the gravity (see Figure 2.2). In [Schüthe et al. 2016] we already mentioned that neglecting the Coriolis and link friction gives still a good model.

The dynamics function can then be retrieved translating Equations (3.2) and (3.3) into a state space representation of the form $\dot{\boldsymbol{x}} = f_\mathrm{dyn}(\boldsymbol{x}, \boldsymbol{u})$ using the state of link and motor positions and velocities

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{q} & \dot{\boldsymbol{q}} & \boldsymbol{\theta} & \dot{\boldsymbol{\theta}} \end{pmatrix} \ , \tag{3.4}$$

the input is defined as the motor torque

$$\boldsymbol{u} = \boldsymbol{\tau}_\mathrm{m} \ , \tag{3.5}$$

to get the dynamics

$$f_\mathrm{dyn}(\boldsymbol{x}, \boldsymbol{u}) = \begin{pmatrix} \dot{\boldsymbol{q}} \\ -\boldsymbol{M}(\boldsymbol{q})^{-1}\boldsymbol{\tau}_\mathrm{c} \\ \dot{\boldsymbol{\theta}} \\ \mathrm{diag}\left(\boldsymbol{b}_\mathrm{m}\right)^{-1}\left(\boldsymbol{\tau}_\mathrm{c} + \boldsymbol{u} - \boldsymbol{\tau}_\mathrm{fm}\right) \end{pmatrix} \ . \tag{3.6}$$

3.2.1. Motor Torque

The motor torque can be extracted from the basic Equation (2.3) and the fact that torque and current are directly coupled by a factor of k_τ, i. e.

$$\tau_\mathrm{ms} = k_\tau I_\mathrm{m} \ . \tag{3.7}$$

Let us plug Equation (2.3) into (3.7)

$$\tau_{\mathrm{ms}} = k_\tau \frac{U_{\mathrm{m}} \left(p - k_{\mathrm{pv}} \dot{\theta}_{\mathrm{m}} \right)}{R_{\mathrm{m}}} \tag{3.8}$$

and rewrite it to get the torque in dependency of pwm p and velocity $\dot{\theta}_{\mathrm{m}}$

$$\tau_{\mathrm{ms}} = \frac{k_\tau}{R_{\mathrm{m}}} U_{\max} p - \frac{k_\tau k_{\mathrm{pv}} U_{\max}}{R_{\mathrm{m}}} \dot{\theta}_{\mathrm{m}} \ . \tag{3.9}$$

In the last step we need to transform the torque to the link side, i. e.

$$\tau_{\mathrm{m}} = r_{\mathrm{g}} \tau_{\mathrm{ms}} \tag{3.10}$$

$$= \underbrace{\frac{r_{\mathrm{g}} k_\tau}{R_{\mathrm{m}}}}_{k_{\mathrm{pwm}}} U_{\max} p - \underbrace{\frac{r_{\mathrm{g}}^2 k_\tau k_{\mathrm{pv}} U_{\max}}{R_{\mathrm{m}}}}_{k_{\mathrm{mi}}} \dot{\theta} \ . \tag{3.11}$$

Finally, we can write it into a matrix form to hold for our three joints.

$$\boldsymbol{\tau}_{\mathrm{m}} = \mathrm{diag}\left(k_{\mathrm{pwm}}\right) U_{\max} \boldsymbol{p} - \mathrm{diag}\left(k_{\mathrm{mi}}\right) \dot{\boldsymbol{\theta}} \tag{3.12}$$

3.2.2. Motor Friction

Friction appears in several forms. There is static friction, i. e. the torque needed to start moving which is higher than the one needed to maintain moving. Another is viscous friction – which increases proportional to velocity, i. e. the faster the motor gets the higher is its friction [Olsson et al. 1998].

The most interesting friction for us is the Coulomb or kinetic friction, where the friction is constant over all velocities. We have to deal with this during the operation of the robot, as it is mostly in motion. The other two can be neglected.

The kinetic friction is a signum function (sgn). The friction is described by

$$\boldsymbol{\tau}_{\mathrm{fm}} = \mathrm{diag}(\boldsymbol{\mu}_{\mathrm{fm}}) \, \mathrm{sgn}(\dot{\boldsymbol{\theta}}) \ . \tag{3.13}$$

This function has the disadvantage of being discontinuous which leads us to a simplification of the friction by approximating the signum by a sigmoid function so that

$$\boldsymbol{\tau}_{\mathrm{fm}} = 2 \, \mathrm{diag}(\boldsymbol{\mu}_{\mathrm{fm}}) \begin{pmatrix} \frac{1}{1+\exp(-400\dot{\theta}_1)} - 0.5 \\ \frac{1}{1+\exp(-400\dot{\theta}_2)} - 0.5 \\ \frac{1}{1+\exp(-400\dot{\theta}_3)} - 0.5 \end{pmatrix} \ . \tag{3.14}$$

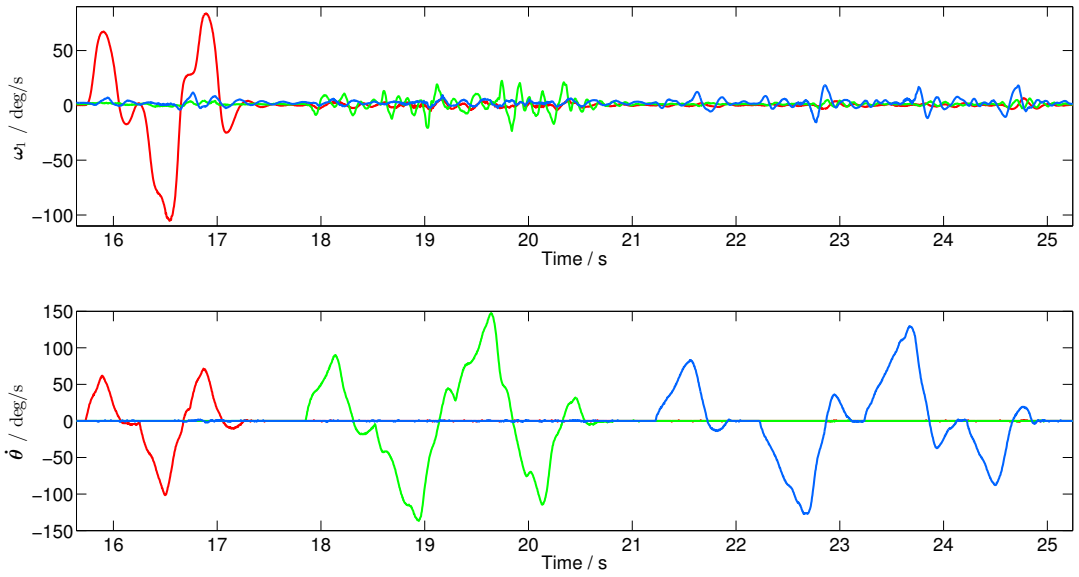


Figure 3.1.: Body one gyroscope data compared to joint velocity. Motions of pitch and roll axes are also detected due to the elasticity in the yaw bearing. Ideally only the red yaw motion would be detected in the gyroscope, as the IMU is placed before the pith and roll joint. For comparability, the gyroscope data is transformed to joint representation.

3.3. Extension for Flexible Bearing

The kinematics and dynamics described until here expect that there is no elasticity in the bearing, i. e. the bearing is perfect. In most applications an elasticity would not even be noticed and it could be neglected. For our robotic system we recognized a significant elasticity in the yaw bearing which we have to consider. The elasticity we see is in the direction of the roll and pitch axes. Both axes stimulate the bearing by their motions. Figure 3.1 illustrates this behavior by measuring the rotational velocities of the first body with IMU I1. If the bearing would be perfect – i. e. it only moves around its rotational axis – the gyroscope will only detect motions of the yaw axis, as the IMU is placed before the pith and roll joint. However, it can be seen that the robot's motion excides vibrations.

The flexibility in the bearing can be simulated and added to the dynamics as two extra revolute joints placed before the yaw axis. The first bearing joint JB1 coincides with the pitch coordinate system J2, whereas the second bearing joint JB2 coincides with the roll coordinate system J3. Both bearing coordinate origins coincide with the origin of the yaw coordinate system. The changes to the coordinate transformations are given in Appendix A.2. Thus, the kinematics for the modified model is

$$f_{\text{kin}}(\boldsymbol{x}) = \begin{pmatrix} 233.4\mathrm{s}_{\mathrm{b}1}\mathrm{c}_{\mathrm{b}2}+1010.069\{\mathrm{c}_3[\mathrm{s}_2(\mathrm{c}_{\mathrm{b}1}\mathrm{c}_1-\mathrm{s}_{\mathrm{b}1}\mathrm{s}_{\mathrm{b}2}\mathrm{s}_1)+\mathrm{s}_{\mathrm{b}1}\mathrm{c}_{\mathrm{b}2}\mathrm{c}_2]-\mathrm{s}_3(\mathrm{c}_{\mathrm{b}1}\mathrm{s}_1+\mathrm{s}_{\mathrm{b}1}\mathrm{s}_{\mathrm{b}2}\mathrm{c}_1)\} \\ 233.4\mathrm{s}_{\mathrm{b}2}+1010.069[\mathrm{c}_3(\mathrm{s}_{\mathrm{b}2}\mathrm{c}_2+\mathrm{c}_{\mathrm{b}2}\mathrm{s}_1\mathrm{s}_2)+\mathrm{c}_{\mathrm{b}2}\mathrm{c}_1\mathrm{s}_3] \\ 233.4\mathrm{c}_{\mathrm{b}1}\mathrm{c}_{\mathrm{b}2}+1010.069\{\mathrm{s}_3(\mathrm{s}_{\mathrm{b}1}\mathrm{s}_1-\mathrm{c}_{\mathrm{b}1}\mathrm{s}_{\mathrm{b}2}\mathrm{c}_1)-\mathrm{c}_3[\mathrm{s}_2(\mathrm{s}_{\mathrm{b}1}\mathrm{c}_1+\mathrm{c}_{\mathrm{b}1}\mathrm{s}_{\mathrm{b}2}\mathrm{s}_2)-\mathrm{c}_{\mathrm{b}1}\mathrm{c}_{\mathrm{b}2}\mathrm{c}_2]\}+779.8 \end{pmatrix} . \quad (3.15)$$

Where $\mathrm{c}_{\mathrm{b}i} = \cos(q_{\mathrm{b}i})$ and $\mathrm{s}_{\mathrm{b}i} = \sin(q_{\mathrm{b}i})$ are for the bearings coordinates one and two.

We can also model the bearing joints as a spring-damper-system like in Equation (2.2), only with the modification that the motor positions and velocities are zero. That is the bearing joints are of course unactuated. The bearing is coupled to the robotic system only via the inertia. Let us define the concatenation of bearing and link positions to

$$\boldsymbol{q}_{\mathrm{fb}} = \begin{pmatrix} q_{\mathrm{b}1} & q_{\mathrm{b}2} & \boldsymbol{q}^T \end{pmatrix}^T \tag{3.16}$$

and its velocities to

$$\dot{\boldsymbol{q}}_{\mathrm{fb}} = \begin{pmatrix} \dot{q}_{\mathrm{b}1} & \dot{q}_{\mathrm{b}2} & \dot{\boldsymbol{q}}^T \end{pmatrix}^T \quad , \tag{3.17}$$

then the coupling torque becomes

$$\boldsymbol{\tau}_{\mathrm{cfb}} = \mathrm{diag} \underbrace{\begin{pmatrix} \boldsymbol{k}_{\mathrm{b}} \\ \boldsymbol{k}_{\mathrm{s}} \end{pmatrix}}_{\boldsymbol{k}_{\mathrm{sfb}}} \begin{pmatrix} \boldsymbol{q}_{\mathrm{b}} \\ \boldsymbol{q} - \boldsymbol{\theta} \end{pmatrix} + \mathrm{diag} \underbrace{\begin{pmatrix} \boldsymbol{d}_{\mathrm{b}} \\ \boldsymbol{d}_{\mathrm{s}} \end{pmatrix}}_{\boldsymbol{d}_{\mathrm{sfb}}} \begin{pmatrix} \dot{\boldsymbol{q}}_{\mathrm{b}} \\ \dot{\boldsymbol{q}} - \dot{\boldsymbol{\theta}} \end{pmatrix} \quad , \tag{3.18}$$

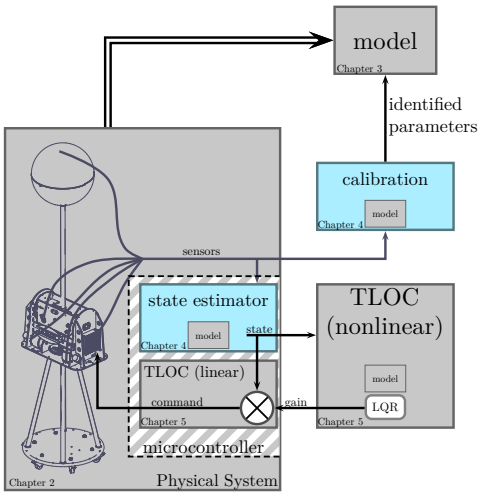
with the spring and damping constants added for the flexible bearing. The extended dynamics is structurally close to the previously defined dynamics, except that for the coupling torque in the motor only the last three vector entries are used. These entries equal exactly the previously defined coupling torque, i. e. $\boldsymbol{\tau}_{\mathrm{cfb},3\ldots5} \equiv \boldsymbol{\tau}_{\mathrm{c}}$. This leads to the flexible bearing state vector

$$\boldsymbol{x}_{\mathrm{fb}} = \begin{pmatrix} \boldsymbol{q}_{\mathrm{fb}} & \dot{\boldsymbol{q}}_{\mathrm{fb}} & \boldsymbol{\theta} & \dot{\boldsymbol{\theta}} \end{pmatrix} \quad , \tag{3.19}$$

and the flexible bearing dynamics with the extended link inertia $\boldsymbol{M}_{\mathrm{fb}}(\boldsymbol{q}_{\mathbf{fb}})$

$$f_{\mathrm{fb\,dyn}}(\boldsymbol{x}_{\mathrm{fb}}, \boldsymbol{u}) = \begin{pmatrix} \dot{\boldsymbol{q}}_{\mathrm{fb}} \\ -\boldsymbol{M}_{\mathrm{fb}}(\boldsymbol{q}_{\mathbf{fb}})^{-1} \boldsymbol{\tau}_{\mathrm{cfb}} \\ \dot{\boldsymbol{\theta}} \\ \mathrm{diag}\left(\boldsymbol{b}_{\mathrm{m}}\right)^{-1} \left(\boldsymbol{\tau}_{\mathrm{c}} + \boldsymbol{u} - \boldsymbol{\tau}_{\mathrm{fm}}\right) \end{pmatrix} \quad . \tag{3.20}$$

In this new formulation it can be seen directly, that the flexible bearing only acts on the link side and the joints recognizing a motion of the bearing by the extended link inertia $\boldsymbol{M}_{\mathrm{fb}}(\boldsymbol{q}_{\mathbf{fb}})$. Also the bearing is stimulated by the joints only through the extended link inertia. Thus, a force transfer takes place in both directions.



4

System Calibration and State Estimation

Knowing the system is fundamental to work with it, i. e. having the knowledge of the system's behavior and its parameters to handle and predict it. Moreover, a good fitting model is essential for our Task Level Optimal Controller (TLOC), because the computations are based on it. In the previous chapters we already got to know the physical structure of the robot with its dynamics and kinematics, its sensors, the parameters for flexible joints, and for the motors. In the first part of this chapter we will see how these parameters can be calibrated offline to obtain dynamics parameters for the model. Section 4.2 is a summary of the calibration paper [Schüthe et al. 2016]. The extension of the flexible bearing model calibration is given in Section 4.3.

The second part of this chapter (Section 4.4) deals with the online state estimation. The actual state x of the robot is needed for our controller, i. e. current positions and velocities of motors ($\boldsymbol{\theta}$, $\dot{\boldsymbol{\theta}}$) and links ($\boldsymbol{q}$, $\dot{\boldsymbol{q}}$). This state is not measurable directly by the sensors. The only state value which can be measured directly is the position of the motors. The motor velocity can be estimated by the given motor position. To estimate the link values we use indirect link velocity measurements of the gyroscopes. To realize the linear acting part of the controller the state estimator has to be implemented on the microcontroller (see Fig. 1.2 on page 3).

We start with an overview of the related work in the field of dynamics calibration and state estimation.

4.1. Related Work

The related work is split into two subsections. The first describes calibration procedures and the second state estimation methods.

4.1.1. Calibration

Basically, there are two methods of identifying the dynamics parameters – on-line and off-line methods – which are summarized in [Wu et al. 2010]. The on-line methods run parallel to adapt the parameters during the process. Thus, also changes in the system will be recognized. Adaptive control algorithms are an example for this [Slotine et al. 1987]. Another method is the identification by neural networks, where the weights represent the parameters and are approached in real-time [Narendra et al. 1990].

The off-line methods can be distinguished into three areas: (1.) Physical experiments: E. g. measure inertia with its respective center of gravity (COG) and the mass of each isolated link. (2.) Using a computer aided design (CAD) software for distances, inertias and COGs. This is restricted to modeled parts, no physical parameters can be obtained, like friction. (3.) Minimizing the difference of estimated data and real data by adjusting the model parameters. We used a combination of (2.) and (3.) for dynamics calibration [Schüthe et al. 2016].

The calibration of industrial robots is investigated in [Grotjahn et al. 2001] and [Vuong et al. 2009]. In both works a rigid body model is assumed. Dealing with flexible joints has been discussed in [Kurze et al. 2008; Moberg 2010], where also the friction model was estimated.

An uncommon method to identify the stiffness of joints using bandpass filtering is presented in [Pham et al. 2001]. But the model ignores the spring damping and identifies each joint independently.

4.1.2. State Estimation

Many online systems are based on a Kalman Filter (KF) given in three basic forms: The KF [Kalman 1960], the Extended Kalman Filter (EKF) [Hoshiya et al. 1984] and the Unscented Kalman Filter (UKF) [Wan et al. 2000]. Where the KF acts only on linear systems, the EKF is an extension to nonlinear systems by linearization, and the UKF is a nonlinear filter method.

Using IMUs – i. e. accelerometers and gyroscopes – to estimate the flexible robots state and use it for control has been shown in [Cheng et al. 2010; Staufer et al. 2012]. But, the state is estimated by both sensors.

Quigley et al. presented a control algorithm using states which were estimated from accelerometer measurements passed through an EKF. Here each couple of joints needs at least one IMU [Quigley et al. 2010]. Estimating the link positions and velocities by accelerometer data is done in [Luca et al. 2007].

A Kinematic Kalman filter (KKF) was presented in [Chen et al. 2014] to determine the EOF position and velocity. An IMU accelerometer is used for link position estimate together with a camera system which senses a 3D measurement system for getting the ground truth. Using a stereo camera system and a gyroscope in addition to the accelerometer and extend the KKF to a multidimensional filter gives the EOF position in [Jeon et al. 2009].

After our paper has been accepted, Xinjilefu et al. published a paper where only gyroscopes are used to estimate the joint angular velocity on a humanoids right and left foot and knee [Xinjilefu et al. 2016]. This approach is very similar to our approach and shows that we are at the state of the art.

4.2. The Calibration Procedure

The calibration of "Doggy" is based on the Sparse Least Squares on Manifolds (SLoM) toolkit [Hertzberg et al. 2012]. In the published paper [Schüthe et al. 2016] we calibrated the system using the dynamics Equation (3.6) without the flexible bearing, which we only summarize in this section.

Finding the parameters that are best explaining the sensory data of a calibration motion is the main goal of the calibration. "I. e. we search for the parameters $\boldsymbol{\vartheta}$ which result in the least squared difference of the actual measurements $[\boldsymbol{Z}]$ and the measurements predicted from the parameters $[\hat{\boldsymbol{Z}}]$. Formally, we search the least-squares estimate $\hat{\boldsymbol{\vartheta}}$" [Schüthe et al. 2016, p.339]. The measurements are combined in the error function $F(\boldsymbol{Z}, \boldsymbol{\vartheta})$.

$$\hat{\boldsymbol{\vartheta}} = \underset{\boldsymbol{\vartheta}}{\operatorname{argmin}} \frac{1}{2} \|F(\boldsymbol{Z}, \boldsymbol{\vartheta})\|^2_{\boldsymbol{\Sigma}} \tag{4.1}$$

We despite between time-invariant parameters $\boldsymbol{\vartheta}_{\text{calib}}$ and time-dependent state parameters $\boldsymbol{\vartheta}_{\text{state},n} = \boldsymbol{x}_n$. The latter are needed to formulate expected measurements in F. Where n denotes the discrete time step. The stacked calibration parameter vector[1] is

$$\boldsymbol{\vartheta}_{\text{calib}} = \begin{pmatrix} k_{\text{pwm}} & k_{\text{mi}} & \boldsymbol{b}_{\text{m}}^T & \boldsymbol{\mu}_{\text{fm}}^T & \boldsymbol{k}_{\text{s}}^T & \boldsymbol{d}_{\text{s}}^T & \boldsymbol{\omega}_{1,0}^T & \boldsymbol{\omega}_{2,0}^T & {}^{\text{I1}}_{\text{J1}}\boldsymbol{R}^T & {}^{\text{I2}}_{\text{EOF}}\boldsymbol{R}^T \end{pmatrix} , \tag{4.2}$$

where ${}^{\text{I1}}_{\text{J1}}\boldsymbol{R}$ and ${}^{\text{I2}}_{\text{EOF}}\boldsymbol{R}$ are the rotation matrices from Joint one to IMU one and from end-effector to the heads IMU. The rotation matrices are inserted to get the exact orientation which might diverge from the CAD-model. The bias' $\boldsymbol{\omega}_{i,0}$ is assumed to be constant over the calibration time. The resultant parameter vector is

$$\boldsymbol{\vartheta} = \begin{pmatrix} \boldsymbol{\vartheta}_{\text{calib}}^T & \boldsymbol{\vartheta}_{\text{state},n}^T & \boldsymbol{\vartheta}_{\text{state},n+1}^T & \cdots & \boldsymbol{\vartheta}_{\text{state},N}^T \end{pmatrix} . \tag{4.3}$$

The SLoM framework performs the minimization and therefor needs the models structure (Fig. 4.1), namely which measurements depend on which parameters. It is important

[1] Mathematically precise $\boldsymbol{\vartheta}$ must be a set of parameters, but can be written as vector, iff matrices are stacked column by column.

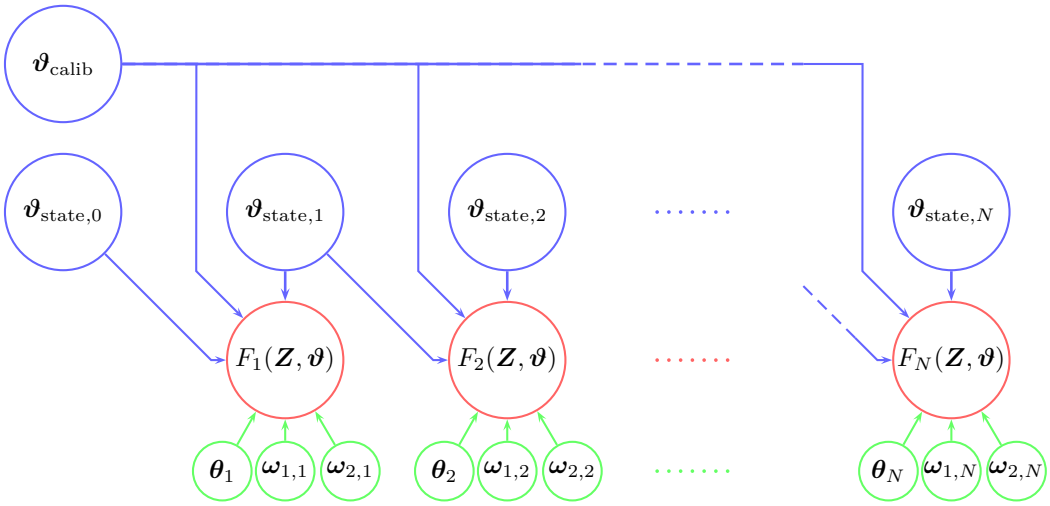


Figure 4.1.: SLoM calibration principle for parameter estimation on "Doggy". Error functions (red) combine measurements (green) and parameters (blue) to obtain the estimated parameters that best explain the measurements.

that the state parameters have influence on the current and the next error function. The state $\boldsymbol{x}_{n+1}$ can be predicted from the previous state and the previous command using the dynamics function with an uncertainty of $\boldsymbol{\Sigma}$ (covariance matrix). This behavior is used to indirectly estimate the parameters of the dynamics. The measurements are needed for the state estimation and orientation of the IMUs.

The parameters calibrated for our model were verified by taking the first state $\boldsymbol{x}_0 = \boldsymbol{\vartheta}_{\text{state},0}$ from SLoM algorithm and predict the whole calibration motion and measurements from that point on. This long term prediction is the toughest task for our model, in its actual use in the state estimator and controller measurements continuously provide fresh information. Whereas the prediction is only based on the provided pwm signals. The result was compared to the measured motion from the robot and is illustrated in our paper. The main point to be noticed is that there are large deviations in the measurements for the yaw and pitch axes, especially in the body IMU $\boldsymbol{\omega}_1$. Figure 4.2 shows this behavior for the roll axis. The deviation between measured (solid) and predicted motor positions (dashed) is due to the expected accumulation of motor velocity errors, because the position results from the integration of the velocity, which is a well known problem when predicting data. The motor velocity fits well to the measured velocity[2]. But, when looking at the IMU measurements, we see larger deviations. The reason is that we neglect the bearing mentioned in Section 3.3. I. e. we neglect motions of yaw and pitch axes in the estimated measurement of $\boldsymbol{\omega}_1$ [see Eqs. (22) and (23) in Schüthe et al. 2016]. The errors for the second IMU are higher when the joint changes direction very quick – to be seen in the very beginning of the figure. Overall, the result looks much

[2]We can not directly measure the velocity, but compute it from the position using the sample time T_s. $\dot{\boldsymbol{\theta}} \simeq \frac{\boldsymbol{\theta}_n - \boldsymbol{\theta}_{n-1}}{T_\text{s}}$ is the average velocity between two samples.

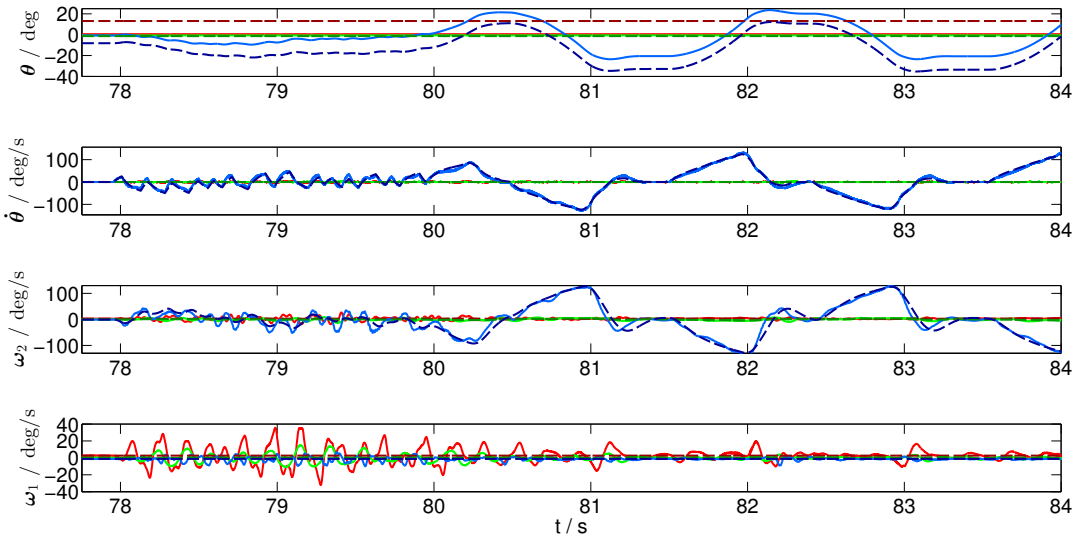


Figure 4.2.: Evaluation of calibrated parameters by comparing measured (solid) and predicted data (dashed). From top to bottom we have the motor position $\boldsymbol{\theta}$ and velocity $\dot{\boldsymbol{\theta}}$, the head $\boldsymbol{\omega}_2$ and body $\boldsymbol{\omega}_1$ IMU. Red, green, and blue denote yaw, pitch, and roll for the motor, x-, y-, and z-axis for the gyroscopes respectively.

better during motions that are not changing direction, as the influence of the bearing is less. A change in direction of yaw and pitch axes stimulates the bearing, which oscillates more the faster the direction change is. This can be seen quite well in the figure. Up to the time of 80 s the motor changes moving direction very quickly, while afterwards the direction changes are smoother and on bigger time intervals. Speaking in frequencies, in the beginning the velocity is a high frequency signal, whereas after the 80 s point the frequency is much lower. And the same is seen for $\boldsymbol{\omega}_1$, first high frequency, then low frequency.

To get a good result in a least-squares sense, SLoM tries to unify the unmodeled bearing flexibility with the spring elasticity between motor and link. Thus, we expected to get better results for the parameters when inserting the flexible bearing into our model, which is explained in the next section.

4.3. Extended Calibration Model

To include the flexible bearing into our model it takes some modifications, already shown for the dynamics in Section 3.3 (Eq. (3.20)). This modification also changes the gyroscopes error and measurement functions [i. e. Eqs. (22)–(29) in Schüthe et al. 2016].

Let us start with Equation (22) and rewrite the error of the bodies gyroscope to

$$F_{\text{gyro1},n} = {}^{\text{I1}}\hat{\boldsymbol{\omega}}_{1,n} - {}^{\text{I1}}\boldsymbol{\omega}_{1,n} \ , \tag{4.4}$$

where n denotes the discrete time step. The measurement estimation is

$$ {}^{\mathrm{I1}}\hat{\boldsymbol{\omega}}_{1,n} = {}^{\mathrm{I1}}_{\mathrm{J1}}\boldsymbol{R}\left({}^{\mathrm{J1}}_{\dot{\boldsymbol{q}}}\boldsymbol{R}\dot{\boldsymbol{q}} + {}^{\mathrm{J1}}_{\dot{\boldsymbol{q}}_\mathrm{b}}\boldsymbol{R}\dot{\boldsymbol{q}}_\mathrm{b}\right) + \boldsymbol{\omega}_{1,0} \ . \tag{4.5} $$

The matrices ${}^{\mathrm{J1}}_{\dot{\boldsymbol{q}}}\boldsymbol{R}$ and ${}^{\mathrm{J1}}_{\dot{\boldsymbol{q}}_\mathrm{b}}\boldsymbol{R}$ are the transforms of link velocities for links $\dot{\boldsymbol{q}}$ and bearings $\dot{\boldsymbol{q}}_\mathrm{b}$ to velocities acting at coordinate system J1. For the second equation, we redefine (26) of the paper to

$$ {}^{\mathrm{I1}}\hat{\boldsymbol{\omega}}_{2,n} = {}^{\mathrm{I2}}_{\mathrm{EOF}}\boldsymbol{R}\left({}^{\mathrm{EOF}}_{\dot{\boldsymbol{q}}}\boldsymbol{R}\dot{\boldsymbol{q}} + {}^{\mathrm{EOF}}_{\dot{\boldsymbol{q}}_\mathrm{b}}\boldsymbol{R}\dot{\boldsymbol{q}}_\mathrm{b}\right) + \boldsymbol{\omega}_{2,0} \ . \tag{4.6} $$

In this case, the error function remains the same and just computes the difference between estimated measurement and the current measurement, i. e.

$$ F_{\mathrm{gyro2},n} = {}^{\mathrm{I2}}\hat{\boldsymbol{\omega}}_2 - {}^{\mathrm{I2}}\boldsymbol{\omega}_{2,n} \ . \tag{4.7} $$

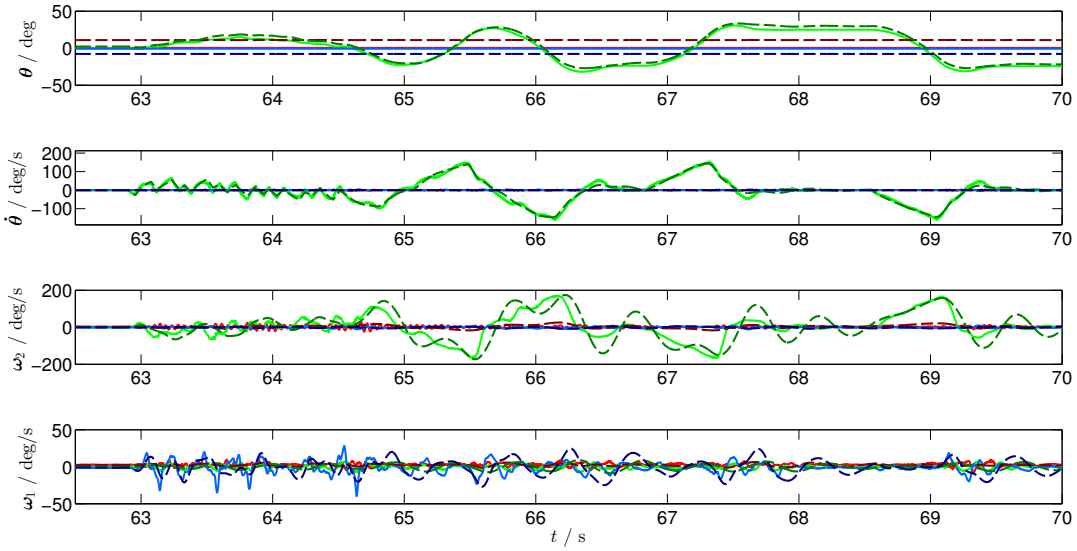
The rotations are given in detail in Appendix A.3.

With these few modifications we are now able to calibrate the dynamics parameters with flexible bearing using the algorithm presented before. We expected the model to be more accurate than the model of the previous section without the bearing. However, it turned out to be worse. The model still fits the motors behavior, but the predicted measurement is not as close as before (Figure 4.3). This is especially true for the pitch joint. But why is it worse? The SLoM algorithm just searches for the parameters that best explain the measurements. That also includes divergence if the measurements are not fitting the model. By including the flexible bearing we might give some degrees of freedom to SLoM that might overfit the model, i. e. the effects could fit either to the bearings spring or the joints spring and motor friction. However, there might be some elasticities in the link included after the flexible bearing that are now packed into the springs of bearing and joints. Including these flexible links into our model would cause an increase of the state space that is unmanageable. But we are going to see in the next section that it is possible to estimate the state using this model.

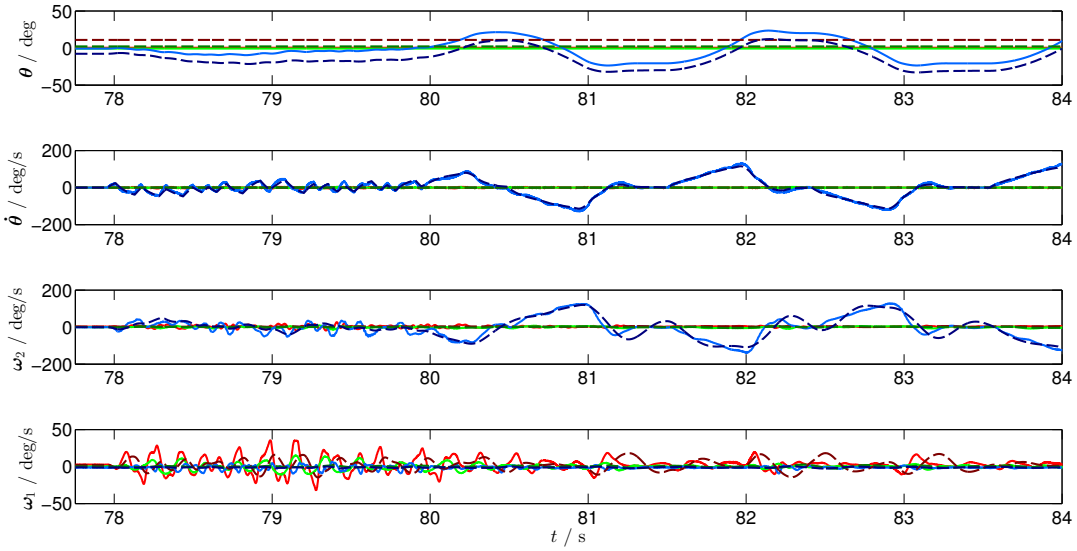
4.4. State Estimation

For the used controller the state as defined in Equation (3.4) (no flexible bearing) or in Equation (3.19) (with flexible bearing) needs to be known. But, we are not able to measure the state vector directly, except for motor positions $\boldsymbol{\theta}$. So, we have to estimate the state given our measurements. This problem is similar to the calibration problem and we will see some equations herein. The idea is that $\boldsymbol{\theta}$ and $\dot{\boldsymbol{\theta}}$ are well observable from the motor encoder. The same holds for $\dot{\boldsymbol{q}}$, which is observable from the gyroscope and hence the short-term behavior of $\boldsymbol{q}$. For the long-term behavior we employ the assumption, that $\boldsymbol{q} - \boldsymbol{\theta}$ is on average close to 0. The gravity terms are compensated by the springs and the acceleration torques cancel out.

We make use of the KF [Kalman 1960] which Kalman presented already in the 60's.



(a) Joint 2 (pitch) predicted motions.



(b) Joint 3 (roll) predicted motions.

Figure 4.3.: Evaluation of calibrated parameters with bearing by comparing measured (solid) and predicted data (dashed). From top to bottom we have the motor positions $\boldsymbol{\theta}$ and velocities $\dot{\boldsymbol{\theta}}$, the head $\boldsymbol{\omega}_2$ and body $\boldsymbol{\omega}_1$ IMU. Red, green, and blue denote yaw, pitch, and roll for the motor, x-, y-, and z-axis for the gyroscopes respectively.

This filter is a linear filter and assumes a linear system. As mentioned before, there also exists the EKF and the UKF for nonlinear systems. Due to computation limitations on the microcontroller, we use the traditional KF and simplify the model.

This filter operates in two steps. The first step predicts the estimated state $\hat{\boldsymbol{x}}_{n|n-1}$ for the next discrete time step n given the measurements of the current time step $n-1$. The same step is done for the covariance matrix $\boldsymbol{\Sigma}_{n|n-1}$, which includes the state process covariance $\boldsymbol{\Sigma}_x$. Both make use of the state transition matrix $\boldsymbol{A}$. Thus, we know the state and its covariance for the next time step.

$$\hat{\boldsymbol{x}}_{n|n-1} = \boldsymbol{A}\hat{\boldsymbol{x}}_{n-1|n-1} \tag{4.8}$$

$$\boldsymbol{\Sigma}_{n|n-1} = \boldsymbol{A}\boldsymbol{\Sigma}_{n-1|n-1}\boldsymbol{A}^T + \boldsymbol{\Sigma}_x \tag{4.9}$$

It follows the update step, where state and covariance are updated given the current measurement $\boldsymbol{Z}_n = \begin{pmatrix}\boldsymbol{\theta}_{\text{meas},n}^T & {}^{\text{I1}}\boldsymbol{\omega}_{1,n}^T & {}^{\text{I2}}\boldsymbol{\omega}_{2,n}^T\end{pmatrix}^T$. To compute state and covariance, the innovation $\boldsymbol{y}_n$ is needed. I. e. the difference of estimated measurement and real measurement

$$\boldsymbol{y}_n = \boldsymbol{Z}_n - \hat{\boldsymbol{Z}}_n \ , \tag{4.10}$$

with the estimated measure

$$\hat{\boldsymbol{Z}}_n = \begin{pmatrix}\hat{\boldsymbol{\theta}}_n^T & {}^{\text{I1}}\hat{\boldsymbol{\omega}}_{1,n}^T & {}^{\text{I2}}\hat{\boldsymbol{\omega}}_{2,n}^T\end{pmatrix}^T \ . \tag{4.11}$$

The estimated measures are already known from the previous section. The estimated state for our current time step is then given by

$$\boldsymbol{K}_n = \boldsymbol{\Sigma}_{n|n-1}\boldsymbol{H}_n^T\left(\boldsymbol{H}_n\boldsymbol{\Sigma}_{n|n-1}\boldsymbol{H}_n^T + \boldsymbol{\Sigma}_{\text{meas}}\right)^{-1} \ , \tag{4.12}$$

$$\hat{\boldsymbol{x}}_{n|n} = \hat{\boldsymbol{x}}_{n|n-1} + \boldsymbol{K}_n\boldsymbol{y}_n \ , \tag{4.13}$$

$$\boldsymbol{\Sigma}_{n|n} = \left(\boldsymbol{I} - \boldsymbol{K}_n\boldsymbol{H}_n\right)\boldsymbol{\Sigma}_{n|n-1} \ . \tag{4.14}$$

Where $\boldsymbol{H}$ maps states to measurements and $\boldsymbol{K}$ is the Kalman gain matrix. The measurement covariance $\boldsymbol{\Sigma}_{\text{meas}}$ denotes the sensors noise. These equations are more detailed in [Kalman 1960], but needed for the filter implementation, to get our state estimate $\hat{\boldsymbol{x}}_{n|n}$ on each sample time.

To obtain the state using the linear KF, we made some assumptions that simplify our dynamics and makes it easier for linearization. First, we neglect the input of our dynamics and treat it as noise. I. e. the covariance $\boldsymbol{\Sigma}_x$ increases for components having the command in it. The second simplification is to neglect the coupling between link and motor $\boldsymbol{\tau}_{\text{c}}$ on the motor dynamics, as the motor velocity can be well estimated by the measured position. And the motor friction can be neglected due to the same argument – velocity is the estimation given by the position. We do the last simplification on the link side, assuming the flexible bearing link inertia $\boldsymbol{M}_{\text{fb}}(\boldsymbol{q}_{\mathbf{fb}})$ to be constant, i. e. $\boldsymbol{M}_0 = \boldsymbol{M}_{\text{fb}}(\boldsymbol{0})$. The covariance $\boldsymbol{\Sigma}_x$ also increases for components that use the link

inertia. Thus, the dynamics for the flexible bearing is

$$f_{\text{fb dyn}}(\boldsymbol{x}_{\text{fb}}, \boldsymbol{u}) = \begin{pmatrix} \dot{\boldsymbol{q}}_{\text{fb}} \\ -\boldsymbol{M}_0^{-1}\boldsymbol{\tau}_{\text{cfb}} \\ \dot{\boldsymbol{\theta}} \\ \boldsymbol{0} \end{pmatrix} = \underbrace{\begin{pmatrix} \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ -\boldsymbol{M}_0^{-1}\boldsymbol{k}_{\text{sfb}} & -\boldsymbol{M}_0^{-1}\boldsymbol{d}_{\text{sfb}} & \boldsymbol{M}_0^{-1}\boldsymbol{k}_{\text{sfb}} & \boldsymbol{M}_0^{-1}\boldsymbol{d}_{\text{sfb}} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}}_{\boldsymbol{A}_{\text{c}}} \boldsymbol{x}_{\text{fb}} \ . \tag{4.15}$$

Where $\boldsymbol{A}_{\text{c}}$ is the continuous state transition matrix, which needs to be discretized to fit for the sample time T_{s}.

$$\boldsymbol{A} = \boldsymbol{I} + T_{\text{s}}\boldsymbol{A}_{\text{c}} \tag{4.16}$$

Only the matrix $\boldsymbol{H}$ is missing yet to compute the estimated state. Basically, this matrix is a concatenation of Equation (20) from the paper and Equations (4.5) and (4.6). The rotation matrices are given by the predicted state, i. e. ${}^{to}_{from}\boldsymbol{R}_n = {}^{to}_{from}\boldsymbol{R}(\hat{\boldsymbol{x}}_{n|n-1})$. Afterwards, we build the derivative of the concatenation at step n with respect to the state $\boldsymbol{x}$. The resultant matrix is

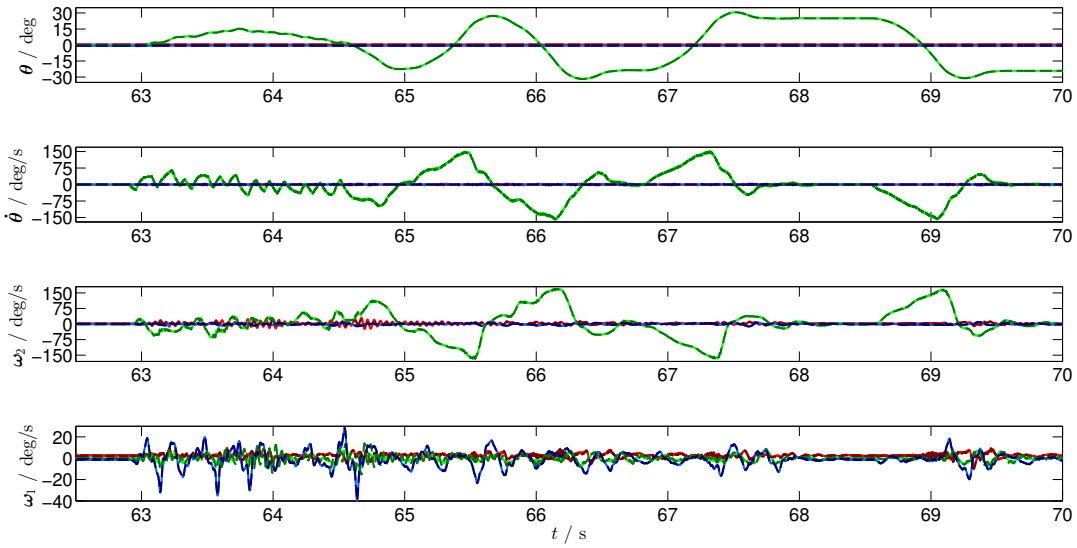
$$\boldsymbol{H}_n = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & {}^{\text{I1}}_{\text{J1}}\boldsymbol{R}\,{}^{\text{J1}}_{\dot{\boldsymbol{q}}_{\text{b}}}\boldsymbol{R}_n & {}^{\text{I1}}_{\text{J1}}\boldsymbol{R}\,{}^{\text{J1}}_{\dot{\boldsymbol{q}}}\boldsymbol{R}_n & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & {}^{\text{I2}}_{\text{EOF}}\boldsymbol{R}\,{}^{\text{EOF}}_{\dot{\boldsymbol{q}}_{\text{b}}}\boldsymbol{R}_n & {}^{\text{I2}}_{\text{EOF}}\boldsymbol{R}\,{}^{\text{EOF}}_{\dot{\boldsymbol{q}}}\boldsymbol{R}_n & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix} \tag{4.17}$$

This implementation has been done on the microcontroller to estimate the state using the flexible bearing modification. We explicitly use this to estimate the bearing positions, too, because the bearing positions influence the camera orientation. This needs to be known for the ball tracking algorithm. To estimate the state on each sample, i. e. running the prediction and the update, the microcontroller needs approximately 1.52 ms[3].

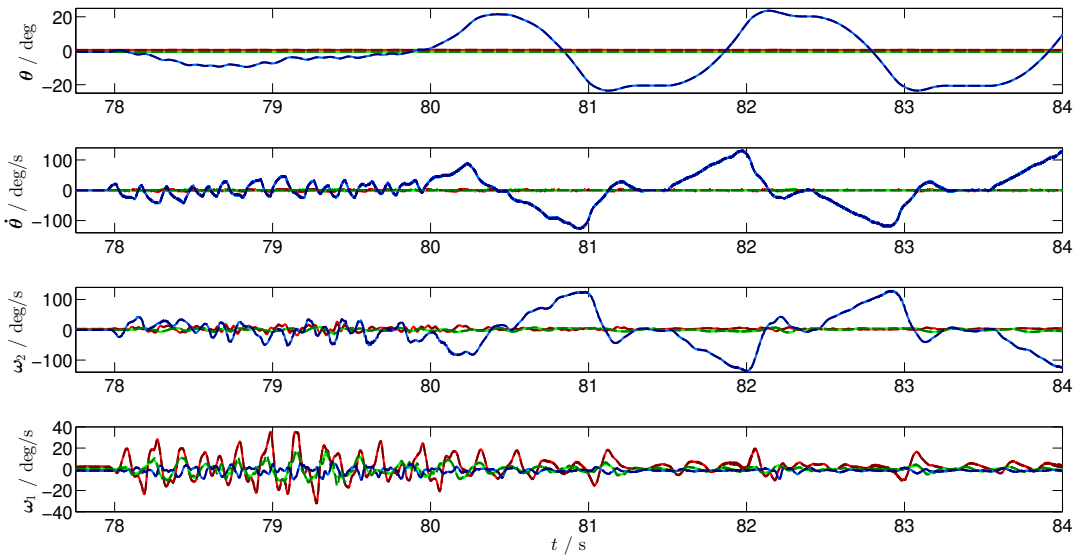
4.4.1. Kalman filter evaluation

For the evaluation we use the same data set that we used for the calibration. We compare it to the measurements and to the states that were found by SLoM with the full model during the calibration process. Here, I just want to pick the two examples of Figure 4.3. In Figure 4.4 we can see that the state transformed to the measurements fits much better to the data then the prediction. There is no drift in the motor positions $\boldsymbol{\theta}$. Moreover, the measurements of the IMU placed on the first body are much more coincident than during the prediction. Both are the result of the measurements going continuously into the estimator. But, we want to see if there is drift on the link position $\boldsymbol{q}$ as result of the integration of the gyroscopes bias'.

[3]We measured the time by toggling a General Purpose Input Output (GPIO) pin to on, when the computation of the filter starts, and off when it has been finished.



(a) Joint 2 (pitch) measurements and estimated measurements.



(b) Joint 3 (roll) measurements and estimated measurements.

Figure 4.4.: Comparison of measured data (solid) and measurements computed from estimated Kalman filter states (dashed). From top to bottom we have the motor positions $\boldsymbol{\theta}$ and velocities $\dot{\boldsymbol{\theta}}$, the head $\boldsymbol{\omega}_2$ and body $\boldsymbol{\omega}_1$ IMU. Red, green, and blue denote yaw, pitch, and roll for the motor, x-, y-, and z-axis for the gyroscopes respectively.

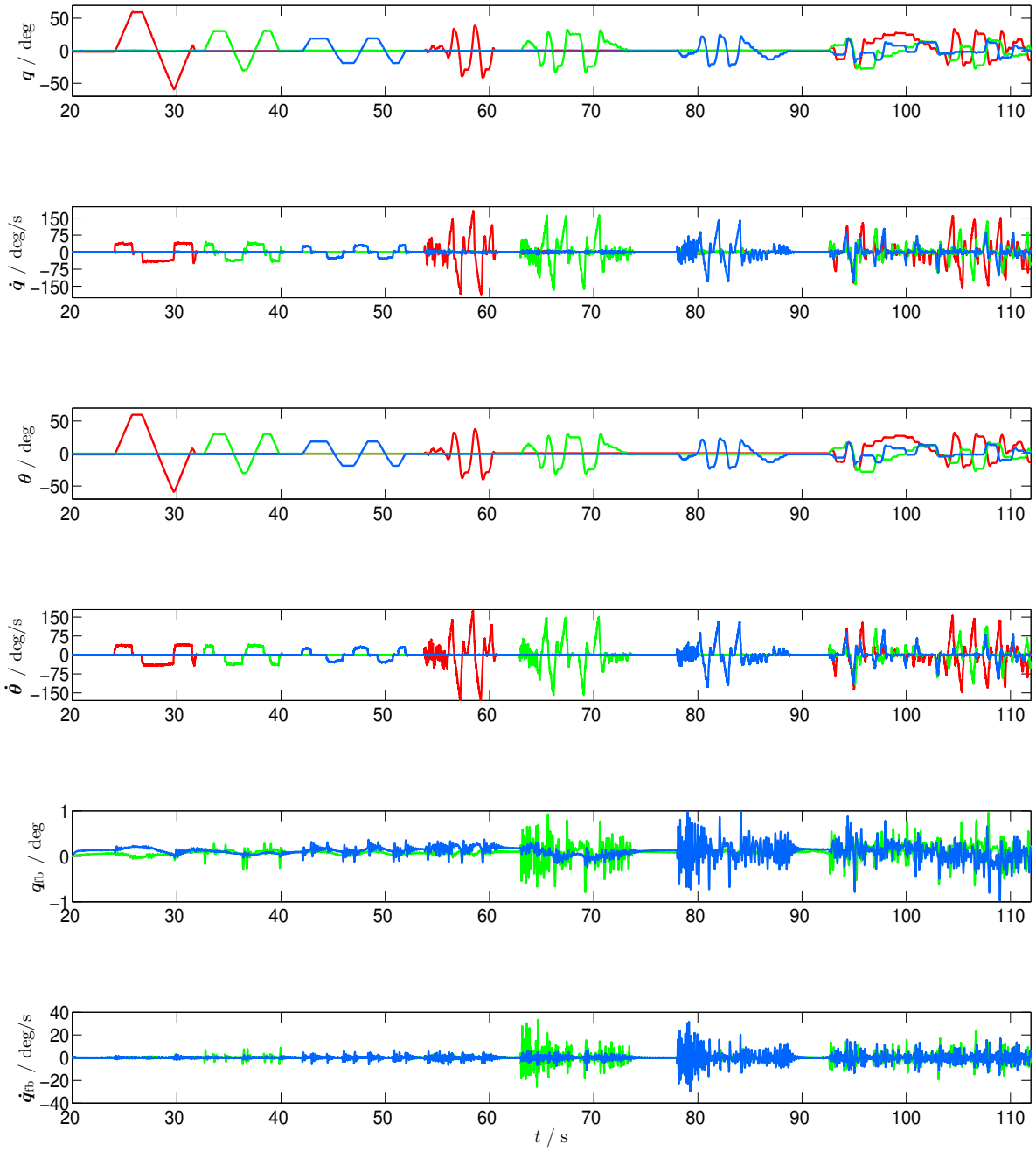


Figure 4.5.: Estimated Kalman filter states on the microcontroller during the calibration motion. From top to bottom we have the motor positions $\boldsymbol{\theta}$ and velocities $\dot{\boldsymbol{\theta}}$, the link positions $\boldsymbol{q}$ and velocities $\dot{\boldsymbol{q}}$, and bearing positions $\boldsymbol{q}_\mathrm{b}$ and velocities $\dot{\boldsymbol{q}}_\mathrm{b}$. Red, green, and blue denote yaw, pitch, and roll axis respectively.

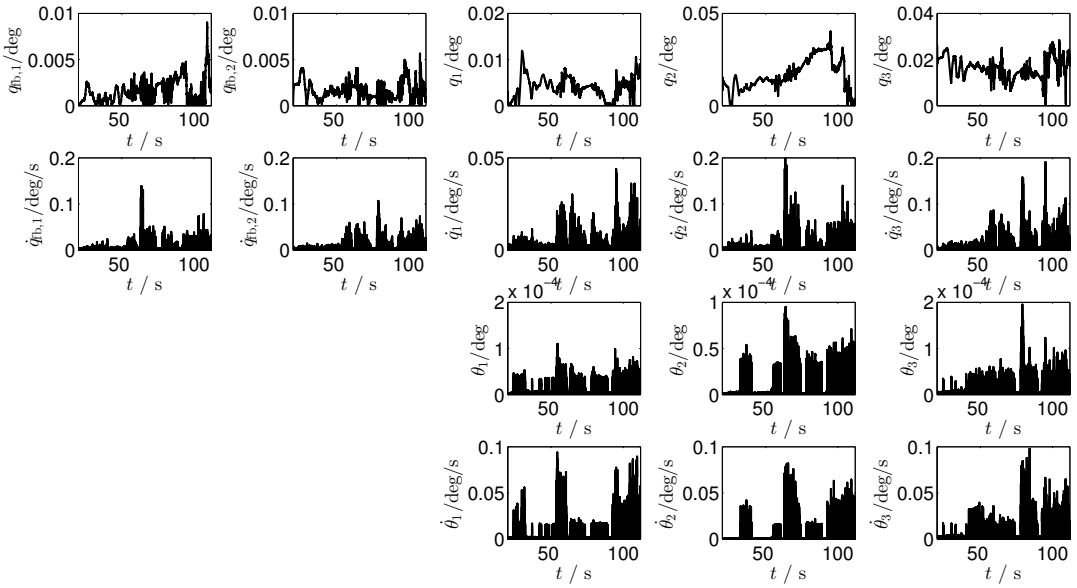


Figure 4.6.: Comparison of estimated states using the offline SLoM calibration and the online running Kalman filter. It is shown the difference of $|\boldsymbol{x}_{\mathrm{SLoM}} - \boldsymbol{x}_{\mathrm{KF}}|$.

Figure 4.5 shows the estimated KF states. In the plot for the link positions there is no drift compared to the motor positions. This is because the joint elasticity links motor and link positions via $\boldsymbol{\tau}_{\mathrm{cfb}}$ (Equation (4.15)). The bearings positions are within a range of $\pm 1°$. During fast motions the deviation to zero increases and faster oscillations are seen compared to motions with limited velocity between 20–53 s.

Unfortunately, in this experiment no ground truth[4] is available, instead we compare the estimated KF states to the estimated SLoM states (Figure 4.6). The comparison to the SLoM states shows only small differences (positions $< 0.041°$; velocities $< 0.2\,° \mathrm{s}^{-1}$). These differences are due to the fact that SLoM has the knowledge about the whole data set $0 \ldots N$ and can optimize it in several steps. Whereas the KF updates are only based on the data of the actual time n, so its knowledge of the measurements is very limited. But, this experiment validates the approximations we made in the KF.

4.4.2. Elastic joint behavior in ball batting motions

We saw that the KF does its job and estimates the state accurately. In this section we run the KF on an example motion inspired by the example of intended ball playing presented in [Schüthe et al. 2015]. Therefore, positions of batting the ball were manually passed to the position controller on the microcontroller to check the states given by the KF. During this movements the link positions and bearing positions were still coupled to the motors position or zero respectively. The overall motion shows the sportive character of the system (Figure 4.7). Let us now get another perspective on the data and see the

[4]Ground truth is an accurate measurement of the same data, here the states.

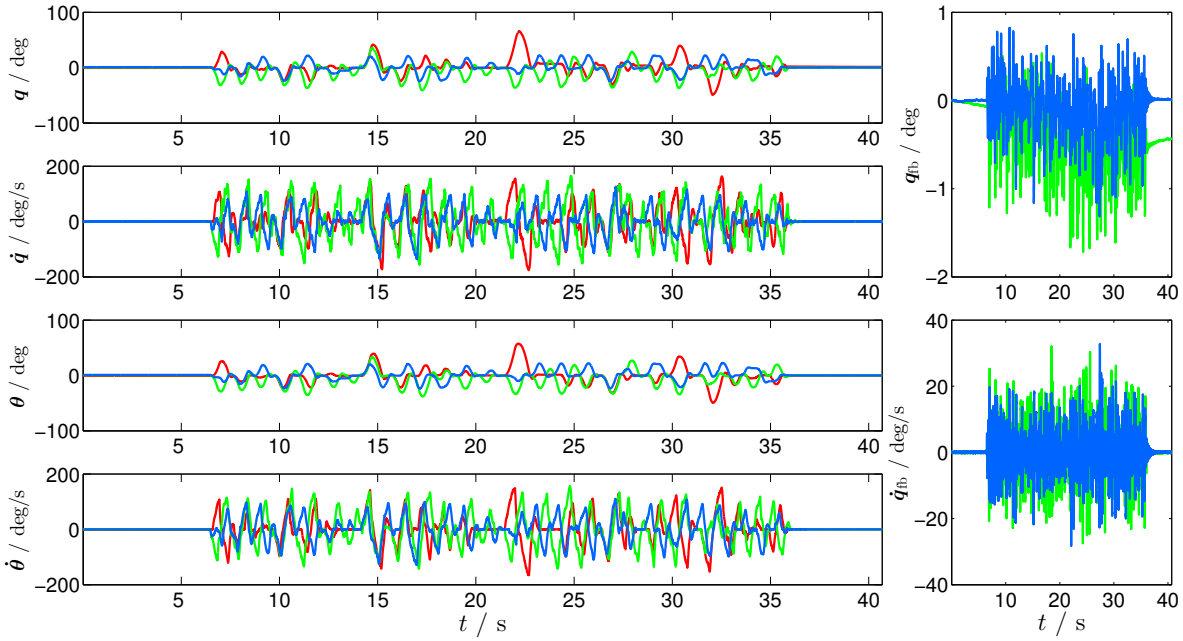


Figure 4.7.: Ball batting motion to check for estimated Kalman filter states. Red, green, and blue denote yaw, pitch, and roll axis respectively.

behavior between motor and link side for positions and velocities.

Due to the flexible joint, the motor and link diverge from another and the link follows the motor. Figure 4.8 shows these characteristics. The flexibility is especially visible in the velocities. The link oscillates around the straighter motor velocity. Moreover, we should note that the decision for dynamics with flexible joint was the right choice. One can neglect the flexibility if the spring stiffness of the coupling is quite stiff. But here we have a system where the flexibility has to be taken into account, as motor and link positions differ in position and velocity significantly. If we neglect the joint flexibility for our batting task, this would lead to a ball trajectory discrepancy that does not hit the ball back to the opponent as desired.

4.5. Summary

The robotic system we presented in the previous chapters was calibrated to get the dynamics parameters and the sensor parameters (i. e. the rotation matrices). A model that predicts the robot's behavior and the measured data has been presented for the two dynamics – with and without the flexible bearing. Surprisingly, the result for the flexible bearing was not as good as the result without the bearing. This is due to deviations in the model and an overfitting of the model. We managed to include the flexible bearing, but may not include some flexibilities caused by the structure after the first joint, which could lead to this effect.

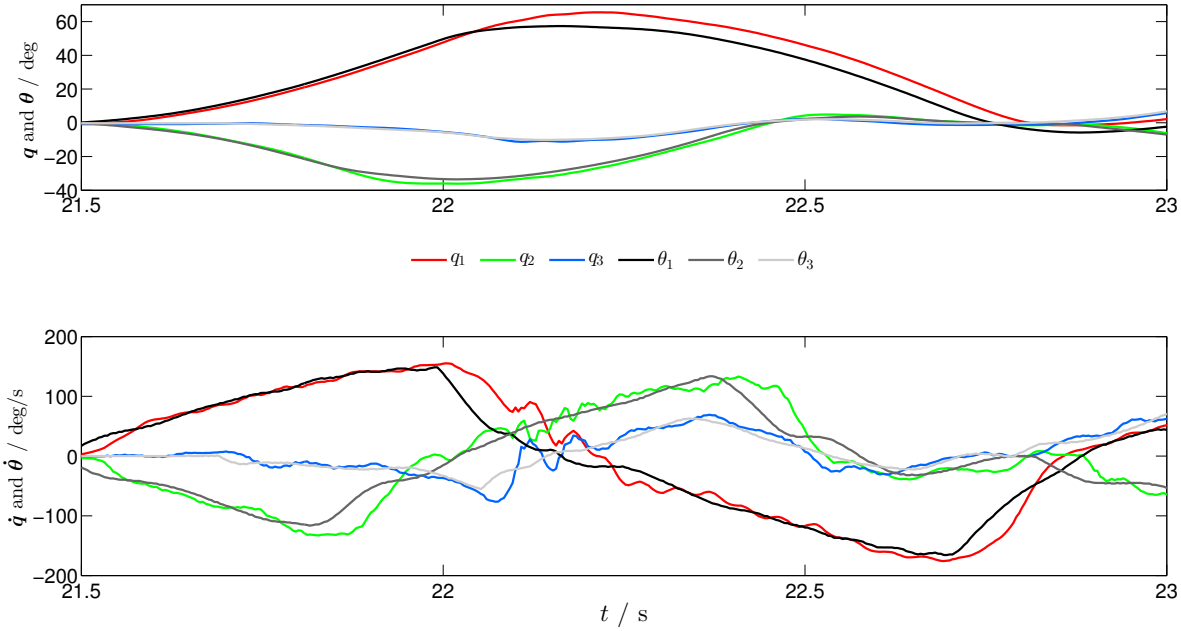
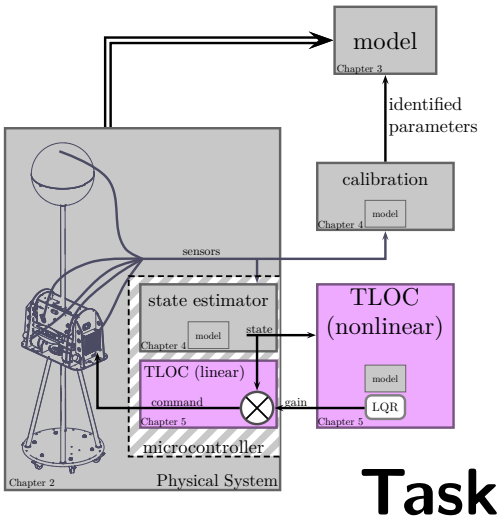


Figure 4.8.: Motor and Link state behavior during a motion.

Based on the dynamics parameters and the flexible bearing model we implemented a Kalman Filter on a microcontroller to estimate the state in real-time. The computation time for the state estimate is about 1.52 ms and within the range of the sample time $T_s = 4$ ms. We have also shown that the estimated state has only small deviations from the measurements and also nearly coincides with the calibrated states of the calibration motion, as no ground truth was available for the comparison. Moreover, a ball batting motion was taken to display that link and motor positions are not drifting away from each other, which would be expected normally for integrating gyroscope data. This holds also for the flexible bearing, which persists close to zero.

We explained that using the flexible joint model is necessary as link and motor positions are significantly different. Additionally, the bearing position could be estimated, which will be essential to provide the camera pose to the ball tracker in the future.

Most notably, we have shown that it is possible to estimate the link positions and velocities using gyroscope data only and, moreover, not having a gyroscope on every link. This, by our knowledge, has not been investigated before. When gyroscopes are used, they are used together with the accelerometers.



5 Task Level Optimal Control

Now we come to the heart of this work — the controller. We aim to answer the question:

Is it possible to define a rebound task for our robot within a controller?

All the work presented until here was investigated to identify the model of the robot needed for the controller and the state estimator. The controller itself is a Task Level Optimal Controller which aim is to control a task, i. e. not controlling the positions given by a trajectory planner. It has the ability to decide which movement is the best for a given situation and thus can directly react to disturbances. Moreover, the name of the controller states that it is optimal. For our task of ball playing that should give a batting motion which is nice and elegant due to its optimality. Having the task defined and controlled by the controller allows to overcome a trajectory planner and having the plant controlled more directly. In addition, we will see that by defining the task the controller is able to actively use the redundancy of the robot to reach the tasks goal. The implementation of the controller is based on the Linear Quadratic Regulator (LQR) algorithm as a framework (Section 5.2).

Afterwards, we describe the generic principle of our controller in Section 5.3. Section 5.4 gives a more detailed view into our specific ball batting controller – the task definition in form of costs. We should mention that the controller has been developed parallel to the robot, i. e. using simulations. Both sections summarize the published papers [Schüthe et al. 2014] and [Schüthe et al. 2015]. In the experimental Section 5.5 we show the controllers' behavior in simulations and in a simplified way on the real robot, to close the gap between simulation and real world.

But we want to start with some related work in the context of optimal control and put our controller into that context.

5.1. Related Work

In [Goretkin et al. 2013] an LQR with finite horizon using Rapidly-exploring Random Trees (RRT)* is used for optimal planning. This is similar to our first approach, where the time stops after the goal has been fulfilled. Mostly, the optimal solution to a control problem is computed, i. e. the input to the plant, [Perez et al. 2012] also computes the control gain explicitly. However, it is not used to control the plant. Another option was presented by [Reist et al. 2010], where the feedback gain of stabilized trajectories is stored in a look up table. All these gains lead to a predefined goal using LQR-Trees.

The iterative refinement using nonlinear dynamics and optimal control is investigated in the iterative LQR [Weiwei Li et al. 2004]. A trajectory gives the linearization points, while the trajectory is optimized by the LQR. This offline mechanism computes the control commands and refines the goal updates given by a tracker. The same method was enhanced to the iterative Linear Quadratic Gaussian (LQG) in [W. Li et al. 2007].

Describing a task to be controlled is mentioned in [Hegyi et al. 2005], where a Model Predictive Controller (MPCs) is used to control the task of having the best traffic flow on a freeway. This is investigated by making an optimal choice between limitations of speed and ramp metering for freeway. A task-level robot learning algorithm is presented in [Aboaf et al. 1988], the robot learns how to throw a ball. This learning is further investigated to juggling in [Aboaf et al. 1989]. Defining a cost function that gives a task based control – in this case a positioning – is shown in [Lenz et al. 2009; Somani et al. 2015]. Here it is used for positioning using a least-square minimization with respect to geometric and kinematic constraints to find the waypoints of a trajectory. Other task specifications are investigated in [De Schutter et al. 2007; Decre et al. 2009] in a general sense. They allow a general task objective function and support inequality constraints in the specification of the task. The command is found by solving a least-squares problem w.r.t. to the constraints. The task is described by the constraints. This is shown on a laser tracing example using a 7 DOF manipulator. An extension up to a specification language and a controller using expression graphs is presented in [Aertbeliën et al. 2014]. The solver translates a specification into a numerical representation which is solved using a quadratic optimization problem including constraints.

Constraints can be equality or inequality constraints, where an equality constraint – such as $\boldsymbol{x} \geq \mathbf{0}$ – is either inactive ($\boldsymbol{x} \geq \mathbf{0}$) or active otherwise ($\boldsymbol{x} < \mathbf{0}$) and then forcing $\boldsymbol{x} = \mathbf{0}$. This optimization leads to an infinite number of decision variables [Scokaert et al. 1998]. Instead we are using soft constraints, i. e. putting the "constraints" into the cost functional leading to a fixed computation time of the LQR algorithm. Moreover, the hard constraints are relaxed as shown for state limitations in [Zeilinger et al. 2014]. Modification of the controller to suboptimal control by the most possible reduction of the horizon to use constraints was investigated in [Johansen et al. 2002]. We need a defined horizon length to get the batting point visible for the controller. Constraining the input command by saturation using quadratic programming was shown in [Mare et al. 2007]. This is done in a similar way in [Goebel 2005], where a saturation function is computed for an LQR using optimal control principle.

5.2. Framework

Our controller used to fulfill the ball batting task is based on the optimal control framework, precisely the Linear Quadratic Regulator principle. This is a well-known controller and can be found in several literature. We describe the principle briefly in this section and refer to [Anderson et al. 2007].

The continuous general optimal control problem can be formulated as follows. Given the initial state $\boldsymbol{x}(t_0)$ and the system $\dot{\boldsymbol{x}} = f_{\text{dyn}}(\boldsymbol{x}, \boldsymbol{u})$, we want to find the optimal control $\boldsymbol{u}^\star(t)$ that minimizes the cost functional for the horizon length T

$$V = \text{cost}_{t_0+T}(\boldsymbol{x}(t_0+T)) + \int_{t_0}^{t_0+T} \text{cost}(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau)\text{d}\tau \ . \tag{5.1}$$

The LQR is based on this principle, too, where the system has to be linear and the cost quadratic. The LQR can be used for continuous and discrete systems. For the discrete LQR the optimal control $\boldsymbol{u}_n^\star$ can be found in deterministic time, by solving the Riccati equation. This is the controller we used for our system as basic framework. An explanation on how the transformation from nonlinear continuous time system to the linear discrete system is done, can be read in our papers. Additionally, the transformation for the costs to the quadratic costs can be found in there. The result is the linear discrete system

$$\boldsymbol{x}_{n+1} = \boldsymbol{A}_n \boldsymbol{x}_n + \boldsymbol{B}_n \boldsymbol{u} \tag{5.2}$$

and the quadratic cost functional

$$J = \boldsymbol{x}_{n_0+N}^T \boldsymbol{Q}_N \boldsymbol{x}_{n_0+N} + \sum_{k=n_0}^{n_0+N} \boldsymbol{u}_k^T \boldsymbol{R}_k \boldsymbol{u}_k + 2\boldsymbol{u}_k^T \boldsymbol{S}_k \boldsymbol{x}_k + \boldsymbol{x}_k^T \boldsymbol{Q}_k \boldsymbol{x}_k \ . \tag{5.3}$$

Where $\boldsymbol{A}$ is the state transmission matrix, $\boldsymbol{B}$ denotes the input matrix, $\boldsymbol{Q} \succeq \boldsymbol{0}$, $\boldsymbol{S} \succeq \boldsymbol{0}$, and $\boldsymbol{R} \succ \boldsymbol{0}$ are cost weights for states, state-command, and commands respectively, which have to be positive semidefinite and positive definite. Minimizing the discrete cost with respect to the command results in the recursive computable feedback gain

$$\boldsymbol{K}_n = -\left(\boldsymbol{R}_n + \boldsymbol{B}_n^T \boldsymbol{P}_{n+1} \boldsymbol{B}_n\right)^{-1} \left(\boldsymbol{S}_n + \boldsymbol{B}_n^T \boldsymbol{P}_{n+1} \boldsymbol{A}_n\right) \ , \tag{5.4}$$

that is used to compute the current optimal control

$$\boldsymbol{u}_n^\star = \boldsymbol{K}_n \boldsymbol{x}_n \ . \tag{5.5}$$

In the equation $\boldsymbol{P}$ denotes the accumulated weight solution of the discrete algebraic Riccati equation (ARE).

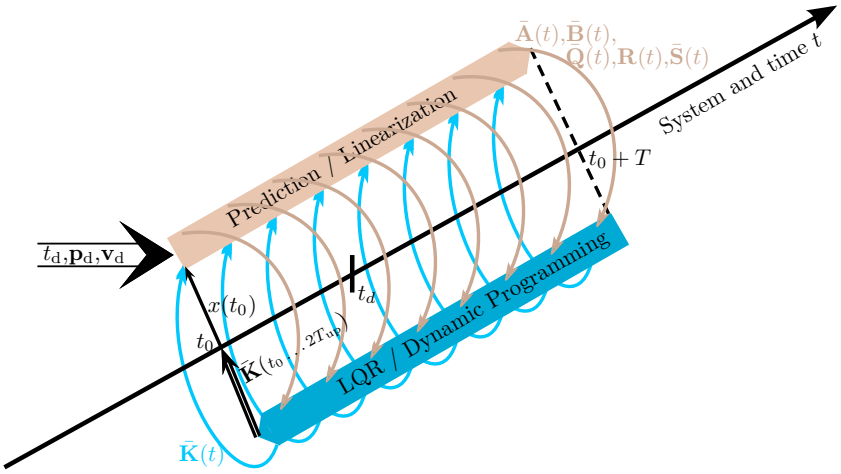


Figure 5.1.: Nonlinear optimization cycle for a given task using an LQR. The black line represents the microcontroller part and its time. The blue and light brown represent the PC part.

5.3. The Principle

The overall concept of the distributed controller was already explained in Section 2.6 on page 14. The nonlinear refinement and optimization runs on the PC, whereas the linear optimal control runs on the microcontroller. Let us take a look into these iterations.

The concept shown in Figure 5.1 consists of two steps made in one iteration cycle. At a given time t_0 our camera system does an update and triggers the microcontroller (black centered line) to pass the actual state $\boldsymbol{x}(t_0)$ to the computer. This state is used to predict the robot's behavior $\boldsymbol{x}_{\mathrm{p},n}$ and $\boldsymbol{u}_{\mathrm{p},n}$ on each sample time n for the given horizon length T using the calibrated model. Based on this prediction the nonlinear system dynamics (Eq. (3.3)) are linearized and discretized to provide the state transition matrix $\bar{\boldsymbol{A}}_n$ and the input matrix $\bar{\boldsymbol{B}}_n$. The bar denotes an affine extension[1] needed to represent the explicit linearization of the system around $\boldsymbol{x}_{\mathrm{p},n}$ and $\boldsymbol{u}_{\mathrm{p},n}$. For a detailed explanation see [Schüthe et al. 2015]. This step is mainly done to take advantage of the LQR principle.

Simultaneously, the cost function is updated based on the prediction. The task is defined in this cost function and is computed for each time step n for the horizon length. The cost is mostly given in a nonlinear way. Thus, the costs have to be linearized, too. As seen in Equation (5.3) the cost has to be quadratic. Therefore, the cost has to be quadratized after the linearization to get the linear weighting matrices ($\bar{\boldsymbol{Q}}_n$, $\bar{\boldsymbol{S}}_n$, and $\boldsymbol{R}_n$) [for details see Schüthe et al. 2015].

Now we reached the end of the light brown arrow in Figure 5.1 having all information to run the dynamic programming part for the LQR (blue line). This part computes the optimal feedback gain $\bar{\boldsymbol{K}}_n$ (Eq. (5.4)).

Note that we explicitly compute the gain matrix and not the optimal commands, which

[1] The affine extended state is $\bar{\boldsymbol{x}} = \binom{\boldsymbol{x}}{1}$ and so the matrices have to fit into this extension, too.

is a main contribution. Because normally an optimal feedback controller computes the optimal command for the actual step only, applies it to the plant and starts over for the next step. The disadvantage is that it must have the result computed in one sample time T_s, which is challenging with dimensional growth of the system. In feedforward control it is done differently. Here, the optimal commands are computed for a given set of times and applied to the plant. However, this method has no chance to adapt to disturbances on the plant.

We implemented a combination of both and apply the feedback gain. This is illustrated in the figure at the end of the blue arrow by transferring the feedback gains to the microcontroller. We do not need to transfer all gains from actual time to the end of the horizon length, because we update in the camera step size T_up. To have a safety margin if the microcontroller misses the camera trigger we transfer gains at least for $2T_\mathrm{up}$. Then the controller on the microcontroller is simply a linear feedback controller computing the command on each sample with $\boldsymbol{u}_n = \bar{\boldsymbol{K}}_n \bar{\boldsymbol{x}}_n$ (Eq. (5.5)).

Let us recapitulate this, as this is an essential part of the controller. The computed gains for the prediction are transferred to the microcontroller – we could say the feedforward gains. But, on the microcontroller the feedforwarded gains are used to compute the actual control command – feedback control. Thus, the controller can act linearly on disturbances, while nonlinear changes due to model discrepancies are handled on every camera update. Then the nonlinear iteration on the PC is triggered to adapt the prediction and the gains – again this is a feedback control.

This step of retriggering the nonlinear iteration closes the loop. Here it should be mentioned that the predicted commands are the result of the previously calculated feedback gains and the predicted states of the actual iteration using Equation (5.5).

The iteration process is summarized in Pseudo Code 1. This illustrates how the computation is done on every camera update and I like to highlight line 7 and 14. The former "throws" the old gains from the last iteration, that have been processed on the microcontroller already, away and appends the same number to the end using the last gain $\bar{\boldsymbol{K}}_{N-1}$ entry N_up times. This needs to be done to have always the same number of feedback gains stored as they are needed to compute the predicted commands. The latter, line 14, transfers the needed feedforward gains to the microcontroller. Additionally, I like to point out line 10. This is the computation of the desired position $\boldsymbol{p}_\mathrm{d}$ and velocity $\boldsymbol{v}_\mathrm{d}$ given a valid ball trajectory, which is computed by the ball tracker. The desired time t_d is when the valid trajectory intersects with the robot's workspace. A concept of how to compute this is given in [Hammer 2011]. These parameters are needed for the ball hitting task which is part of the cost function which we are going to see in the next section now.

5.4. Ball Batting Task Implementation

The principle we discussed in the previous section is now applied to our task. The task we are going to implement for the robot is to hit a recognized ball back to an opponent

Pseudo Code 1: Nonlinear iteration description in the view of the PC.

set $n = 0$
get initial state $\boldsymbol{x}_0 = \boldsymbol{x}(0)$ from microcontroller
Set $\bar{\boldsymbol{K}}_{n\dots N-1} = \boldsymbol{0}$
initialize $\bar{\boldsymbol{A}}_{n\dots N}, \bar{\boldsymbol{B}}_{n\dots N}$ around linearization points $\boldsymbol{x}_0$ and $\boldsymbol{u} = \boldsymbol{0}$
while ∞ **do**
    reset $n = 0$
    remove first N_{up} elements of $\bar{\boldsymbol{K}}$ and append N_{up} times $\bar{\boldsymbol{K}}_{N-1}$, i. e. $\bar{\boldsymbol{K}}_{n\dots N} = \left\{\bar{\boldsymbol{K}}_{N_{\text{up}}\dots N-1} \quad \bar{\boldsymbol{K}}^{(1)}_{N-1} \quad \bar{\boldsymbol{K}}^{(2)}_{N-1} \quad \cdots \quad \bar{\boldsymbol{K}}^{(N_{\text{up}})}_{N-1}\right\}$
    wait for camera update
    get current state $\boldsymbol{x}_0 = \boldsymbol{x}(t)$ from microcontroller
    compute desired head center $n_{\text{d}} = \frac{t_{\text{d}} - t}{T_{\text{s}}}$, $\boldsymbol{v}_{\text{d}}$, $\boldsymbol{p}_{\text{d}}$ from ball trajectory (not part of this thesis)
    predict system behavior $\bar{\boldsymbol{A}}_{n\dots N}$, $\bar{\boldsymbol{B}}_{n\dots N}$, $\bar{\boldsymbol{x}}_{p,n\dots N}$, $\boldsymbol{u}_{p,n\dots N-1}$
    quadratize cost $\bar{\boldsymbol{Q}}_{n\dots N}$, $\bar{\boldsymbol{R}}_{n\dots N-1}$, $\bar{\boldsymbol{S}}_{n\dots N-1}$ for $\bar{\boldsymbol{x}}_{p,n\dots N}$, $\boldsymbol{u}_{p,n\dots N-1}$
    compute gains $\bar{\boldsymbol{K}}_{n\dots N-1}$
    transfer gain matrices $\bar{\boldsymbol{K}}_{n\dots 2N_{\text{up}}}$ to microcontroller
end

player. Therefore, we need to compute the velocity $\boldsymbol{v}_{\text{d}}$ and position $\boldsymbol{p}_{\text{d}}$ that hits the ball back to an opponent. This is done in the ball tracker and also delivers a desired time t_{d}. Thus, the task is *to be at time* t_{d} *with the EOFs coordinate system at position* $\boldsymbol{p}_{\text{d}}$ *having the velocity* $\boldsymbol{v}_{\text{d}}$.

But, we must have in mind that there are other tasks to be fulfilled, which are not obvious on the first view. These are: *Know what to do after hitting the ball!*; *Know your limitations!*; and *Take care about elasticity!*

The task is put into the general optimal control cost function (Eq. (5.1)). We already mentioned that the costs have to be quadratic for the LQR (Eq. (5.3)). How the linearization and quadratization is fulfilled is explained in [Schüthe et al. 2015]. Here, we only want to show what the cost is made of in general. Therefore, have a look at Figure 5.2 where the cost functions are defined (as in [Schüthe et al. 2015]). It can be seen that the cost consists of three parts:

1. Costs that define the main task (above the blue box)

2. Costs that enforce constraints (below the blue box).

3. A terminal cost, which applies to the end of the horizon

The formulas for the costs are given in Figure 5.2 and are explained in the following.

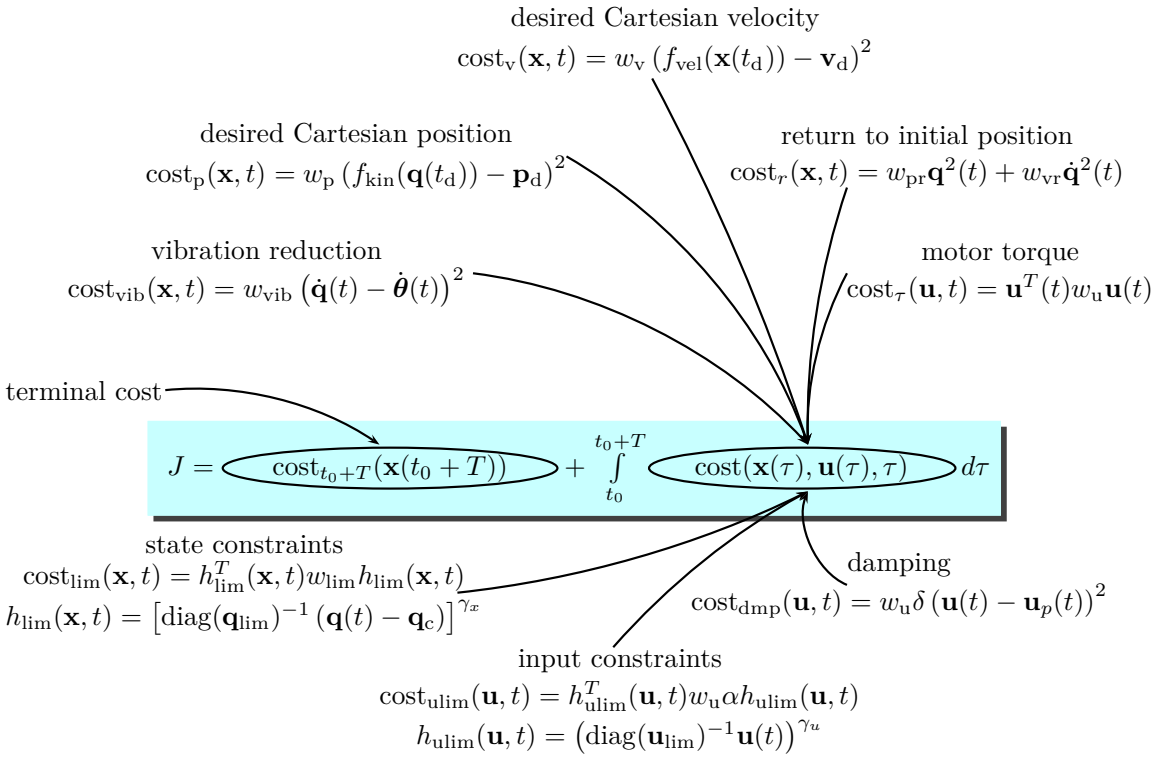


Figure 5.2.: Task of ball hitting plugged into the cost functions. As it can be seen, that also includes subtasks which have to be taken into account. Below the blue box all "constraints" are set. Above the costs to fulfill the main task is shown.

5.4.1. Task costs

The task itself splits into the following three parts that are above the blue box:

Wait and Hit

This part consists of the batting task to be fulfilled at time t_d and the costs for *return to initial position*, i. e. zero for link positions and velocities.

The wait part is the standard case, if there is no valid ball trajectory and the set of $t_\text{d}, \boldsymbol{p}_\text{d}, \boldsymbol{v}_\text{d}$ is empty. The *return to initial position* is active for all times steps $n = 0 \ldots N$ with the weights w_pr and w_vr. This part denotes to stay or go back to the links zeros positions and velocities.

If there is a valid trajectory the hit part with *desired Cartesian position* $\boldsymbol{p}_\text{d}$ and *desired Cartesian velocity* $\boldsymbol{v}_\text{d}$ at time t_d gets active. Then the *return to initial position* costs are only active for times t larger than the desired time added by a time margin t_m, i. e. $\text{cost}_\text{r}(\boldsymbol{x}, t \geq t_\text{d} + t_\text{m})$. This guarantees no conflict between the hit task and the return/wait task. If both would be active or the return starts right after the desired

time, the controller must find a compromise between the two costs. As the goal is to minimize the cost, the controller would decide the middle position/velocity between the two tasks, as this is the lowest cost.

Now, we have to hit the ball at *desired Cartesian position*. This is done by penalizing the difference between the desired Cartesian position $\boldsymbol{p}_\mathrm{d}$ and the state transformed into a Cartesian representation using our kinematics function (3.1). The weight w_p is chosen very high to be accurate and because this cost is only active at the one time-step t_d. The same holds for the *desired Cartesian velocity* $\boldsymbol{v}_\mathrm{d}$, where the difference to the Cartesian velocity is penalized. The function $f_\mathrm{vel}(\boldsymbol{x})$ is just the time derivative of the kinematics (3.1), i. e.

$$f_\mathrm{vel}(\boldsymbol{x}) = \frac{\partial f_\mathrm{kin}(\boldsymbol{x})}{\partial t} = \frac{\partial f_\mathrm{kin}(\boldsymbol{x})}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} \ . \tag{5.6}$$

The weighting factor w_v was chosen factor 10 smaller according to w_p, because if a position error occurs it would have more impact on the precision than a velocity error.

Motor Torque

The cost for the *motor torque* penalizes all torques greater than zero. In other words, the controller should fulfill the batting task with as little torque, i. e. equivalent to motor current, as possible. But, we carefully have to choose the weight w_u. A weight in the dimension of w_v would tell the controller to do nearly nothing, because it is more expensive to generate a torque than moving to the desired position. In contrast a weight close to zero would tell the controller that it could do whatever it wants as long as the batting works, i. e. a torque of infinity would be allowed.

Vibration Reduction

The vibration reduction is only needed to reduce oscillations in the spring. In experiments we found out, that the controller oscillates as a result of the elasticity between link and motor. By inserting the vibration reduction we achieved a smoother result and the oscillations reduced to an acceptable level. This can be handled by penalizing the deviation of link and motor velocity by a weight of w_vib.

This is how the task of hitting the ball is defined using the cost functions. To provide the controller with information on its underlying physical robotic system, we insert soft constraints.

5.4.2. Soft Constraints

We call it soft constraints, because they are acting as a barrier in the costs and could be violated. In contrast to equality or inequality constraints they have no hard limitations. So in the real physical world we always have to check if such limits are exceeded and act on that situations. But the advantage is a deterministic computation time of the

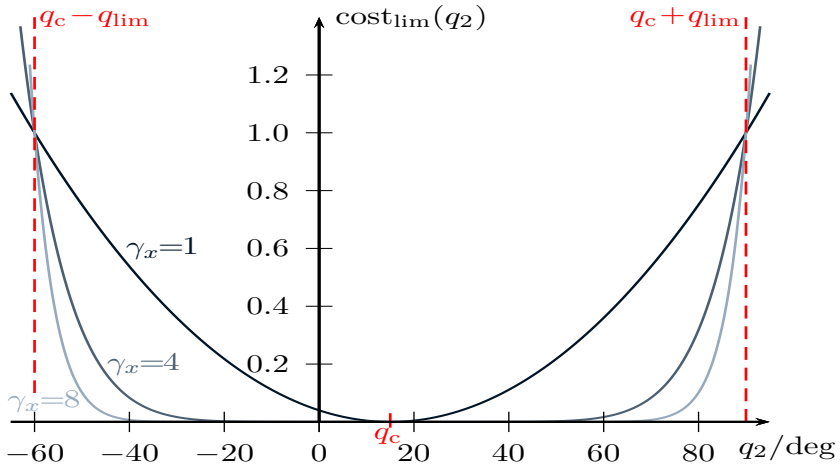


Figure 5.3.: Soft constraint barrier function for the example of the pitch joint. Shown for different parameters. We use $\gamma_x = 8$

.

feedback gains. Moreover, this has the nice effect that the controller gets to know its physical limitations in form of costs and considers these in the optimal feedback gain computation. In the following we are going to explain two physical constraints for the state and the input, and a damping cost function that was additionally inserted due to input constraints.

State Constraints

In the *state constraints* we emphasize the physical limits of the joints, i. e. joint positions limitations $\boldsymbol{q}_{\text{lim}}$. We penalize a deviation from the center between minimum and maximum joint limitation $\boldsymbol{q}_{\text{c}}$. Figure 5.3 illustrates this behavior for the pitch joint. E. g. if the joint angle gets closer to its limits, the cost increases rapidly such that the controller would not like to move further in that direction. This has the nice effect, that the redundancy of the robot is used actively by the controller when positions are close to their limits or the desired position lays outside the limits. The latter one could happen if a desired position $\boldsymbol{p}_{\text{d}}$ is passed which is only reachable by turning the yaw joint and use the pitch joint instead of turning the roll joint over its limits – we will see this in the experiments section. The parameter γ_x is chosen high to have a steep barrier (cf. Fig. 5.3), we use $\gamma_x = 8$.

Input Constraints

The *input constraints* describe the limitations to the commanded torque. This is equivalent to the commanded current and so to the commanded pwm. The maximum torque can be computed from Equations (2.6) and (3.7). The maximum pwm we can set is always $\boldsymbol{p}_{\text{max}} = \boldsymbol{p}_0 \pm 0.2$ independently from the motor velocity. We can plug this into

(3.7) where $\boldsymbol{p}$ is replaced by $\boldsymbol{p}_{\max}$. Then the zero pwm is replaced by Equation (2.4). Following the steps of Equations (3.7)–(3.12) we get the limiting motor torque

$$\boldsymbol{\tau}_{m,\lim} = \pm 0.2 k_{\text{pwm}} U_{\max} \ . \tag{5.7}$$

The factor α increases the cost in total, where γ_u just gives the form of the barrier (smooth or hard).

Moreover, the linearization of the nonlinear input constraints makes the controller oscillate over the nonlinear iteration. Therefore, we inserted a damping term that counters this phenomenon.

Damping

The *damping* penalizes a change of the command over two consecutive iterations. This is similar to the Levenberg-Marquardt method [Press et al. 2002]. It helps convergence by reducing the command change on each iteration. This can be imagined like this: If there is no change limitation and the command gets close to the barrier, the linearization creates a line having a high gradient. The linearized controller thinks that putting a bit less command will lower the cost a lot. So the result is a change to a command far away from the last. The allowed change factor is limited by δ.

5.4.3. Terminal costs

Our terminal costs are a look ahead for the controller. We define that the controller should move to its initial state eventually, i. e. all velocities and positions are going to be zero. So it could do whatever it wants in this last step as everything afterwards is not recognized in the costs. So this cost is more like: *Have in mind to go back at the end*. The cost itself is the same as for *return to initial position*.

5.5. Experiments

Let us now have a look at how the controller presented in [Schüthe et al. 2014, 2015] behaves on the calibrated model and on the robotic system given in Chapter 2. This section is divided into four subsections where we get closer to the real system in every subsection. We start with the simulation presented in [Schüthe et al. 2015] and adapt it to the new model to see if the algorithm still works. Afterwards, we do some tests on the simulation and leave out some information on the model used in the Task Level Optimal Control (TLOC) to test its robustness towards model deviations. In the following subsection the control gains found previously are applied to the simulated plant with the flexible bearing. And finally, we show the feedback gain applied on the hardware.

Before we start I give some information about the settings. Due to comparability reasons we use the model without the bearing, i. e. Equation (3.6) with the parameters denoted in [Schüthe et al. 2016]. When we speak in the following of a “model”, it is

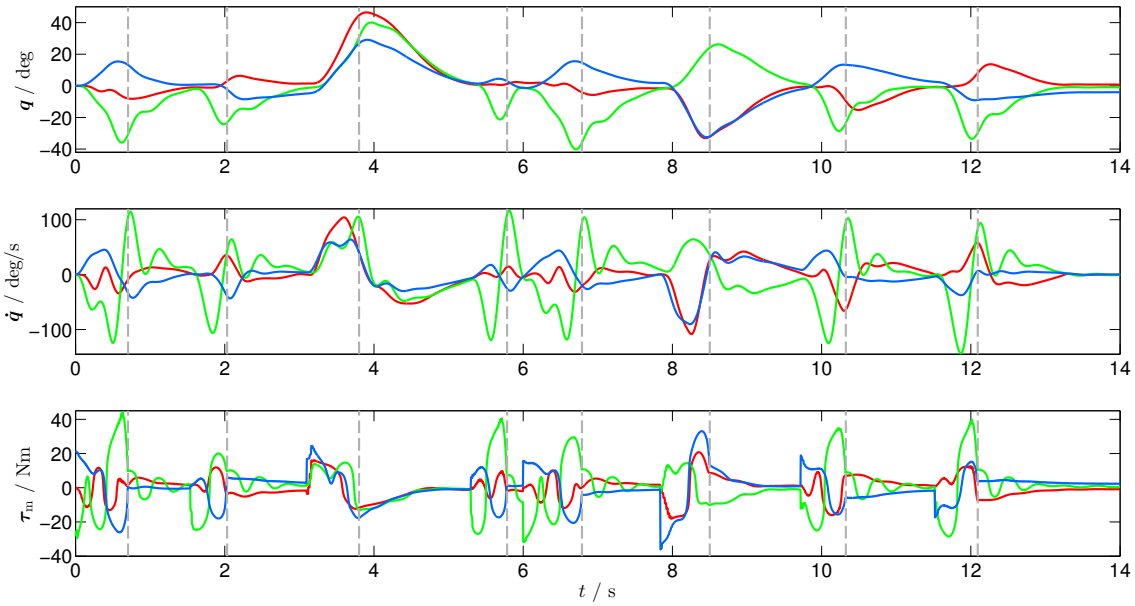


Figure 5.4.: TLOC simulation adapted to the calibrated model. Red, green, blue denote yaw, pitch and roll joint respectively. The gray dashed lines are randomly chosen hit points where position $\boldsymbol{p}_\mathrm{d}$ and $\boldsymbol{v}_\mathrm{d}$ have to be reached.

the model that is part of the optimization in the controller. Whereas the "plant" is the simulated behavior of the robot using also Equation (3.6), but running the simulation using an ODE-solver to have a "time-continuous" simulation. Moreover, the plant has the implemented current limitation discussed in Section 2.3.

5.5.1. TLOC with calibrated model

In this experiment we run our TLOC in a loop for about 14 s choosing random times where a ball should be hit. This includes randomly chosen positions $\boldsymbol{p}_\mathrm{d}$ and velocities $\boldsymbol{v}_\mathrm{d}$ at that time. When the robot has returned to its initial position a new goal has been chosen. The desired time is taken in the interval $t_\mathrm{d} = [0.3\,\mathrm{s},\ 0.7\,\mathrm{s}]$.

In our first runs we experienced an oscillation of the commanded torque. This could be managed by increasing the parameter to penalize vibration $w_\mathrm{vib} = 50\,\mathrm{s\,rad^{-2}}$. Moreover, it turned out that the command limitation does not work well enough. A commanded torque of hundreds of N m could be seen. An adaptation in $\alpha = 500$ gives a solution. These are the only parameters we had to adapt compared to the parameters used in [Schüthe et al. 2015].

Figure 5.4 shows the adapted simulation with desired hit positions and velocities given at the desired gray dashed line. We put this recorded motion in a video[2] for better visualization. In the overall time eight goals where given to the controller. Note

[2] `http://www.informatik.uni-bremen.de/agebv2/downloads/videos/schuetheThesisTLOCSimulation.mp4`

that the limitation of the torque has changed by Equation (5.7). The positions where reached with a mean deviation of $\Delta\bar{\boldsymbol{p}}_\mathrm{d} = \begin{pmatrix} 0.0043 & 0.0031 & 0.0032 \end{pmatrix}^T$ m and the velocities with a deviation of $\Delta\bar{\boldsymbol{v}}_\mathrm{d} = \begin{pmatrix} 0.1221 & 0.0350 & 0.0664 \end{pmatrix}^T \mathrm{m\,s^{-1}}$. Which is accurate enough to hit the ball to the desired position. The highest deviation is in the x direction where the audience is located, this lead to a shorter ball trajectory after the hit. From the view of the audience this means a catch of the ball at height of the knee and not the chest. So it can be handled by the audience. But this deviation has another aspect I would like to highlight.

Finding the perfect parameters for the TLOC is nearly impossible. It is always about finding a compromise for cost weightings. Have a look at the first, the fourth and the latter two hits. At these motions the torque limit is slightly exceeded for the pitch joint (green). This leads to a higher deviation, because on the hardware and also our simulated plant the current is limited. Thus, the plant can not accelerate as fast as the TLOC computed. So the parameters have to be well tuned, and we thought that the parameters set for the TLOC are a good compromise.

Another aspect of the TLOC is the usage of redundancy iff the position can not be reached by just using the pitch and roll joint. Such motions are clearly given at time 3.796 s and 8.496 s. The desired goal is not reachable for the controller with only the pitch and roll joint. It recognizes this by the joint limitations implemented as barrier function in the costs. It reacts as expected and moves the yaw joint. Thus, the joint limits are not exceeded.

By now our plant acts according to the model (except that the former denotes continuous time simulation). In the next subsection we are going to see what happens when the model and the plant differ.

5.5.2. Deviation between plant and model

To control our robot we are using a model based controller, i. e. the model has to be known very precisely. But what happens if the model differs from the plant? We tested this by changing the model and leaving the plant as it is. Here, we change the model to get the "worst case" of control, i. e. the damping constant $d_\mathrm{s} = 0$ for every joint. Moreover, the friction in the model was set to be zero. We tested these settings on the same desired goal ($\boldsymbol{p}_\mathrm{d} = \begin{pmatrix} 0.00 & 0.00 & 2.0233 \end{pmatrix}^T$ m and $\boldsymbol{v}_\mathrm{d} = \begin{pmatrix} 1.7629 & 1.7629 & 0.00 \end{pmatrix}^T \mathrm{m\,s^{-1}}$) and summarized the results in Table 5.1. Each motion is given in Figure 5.5. The best result was achieved by including damping and friction into the model. This was expected, as the plant and the model only differs in the fact that the one is continuous and the other discrete. Depending on the sample time, also differences can occur here, but it should be the closest model describing the plant behavior. Also we see that neglecting the damping is the worst case. We set the damping for the test to zero, i. e. there is no damping. But found out that this leads to a big difference between model and plant, and to the highest deviation to the best model (including damping and friction).

It turned out that neglecting the friction is not a big problem. It is not that accurate reaching its desired goal, but still close to the best solution. However, it is a problem to

Table 5.1.: Comparison of model deviations from the simulated plant. The desired position is $\boldsymbol{p}_\mathrm{d} = (\,0.00\ 0.00\ 2.0233\,)^T\,\mathrm{m}$ and the velocity $\boldsymbol{v}_\mathrm{d} = (\,1.7629\ 1.7629\ 0.00\,)^T\,\mathrm{m\,s^{-1}}$ have to be reached at $t_\mathrm{d} = 0.6\,\mathrm{s}$.

damping	friction	$\boldsymbol{p}(t_\mathrm{d})$ / m	$\boldsymbol{v}(t_\mathrm{d})$ / $\mathrm{m\,s^{-1}}$
no	no	$\begin{pmatrix}0.0041 & 0.0295 & 2.0228\end{pmatrix}^T$	$\begin{pmatrix}1.6079 & 1.5226 & -0.0510\end{pmatrix}^T$
no	yes	$\begin{pmatrix}0.0055 & 0.0250 & 2.0229\end{pmatrix}^T$	$\begin{pmatrix}1.6384 & 1.5354 & -0.0469\end{pmatrix}^T$
yes	no	$\begin{pmatrix}-0.0034 & -0.0042 & 2.0233\end{pmatrix}^T$	$\begin{pmatrix}1.6605 & 1.6458 & 0.0124\end{pmatrix}^T$
yes	yes	$\begin{pmatrix}-0.0011 & -0.0023 & 2.0233\end{pmatrix}^T$	$\begin{pmatrix}1.7112 & 1.7008 & 0.0059\end{pmatrix}^T$

Table 5.2.: Comparison of plant with and without the flexible bearing simulated. The desired position is $\boldsymbol{p}_\mathrm{d} = (\,0.00\ 0.00\ 2.0233\,)^T\,\mathrm{m}$ and the velocity $\boldsymbol{v}_\mathrm{d} = (\,1.7629\ 1.7629\ 0.00\,)^T\,\mathrm{m\,s^{-1}}$ have to be reached at $t_\mathrm{d} = 0.6\,\mathrm{s}$.

bearing in plant	$\boldsymbol{p}(t_\mathrm{d})$ / m	$\boldsymbol{v}(t_\mathrm{d})$ / $\mathrm{m\,s^{-1}}$
not simulated	$\begin{pmatrix}-0.0011 & -0.0023 & 2.0233\end{pmatrix}^T$	$\begin{pmatrix}1.7112 & 1.7008 & 0.0059\end{pmatrix}^T$
simulated	$\begin{pmatrix}-0.0218 & -0.0094 & 2.0230\end{pmatrix}^T$	$\begin{pmatrix}1.5557 & 1.6203 & 0.0487\end{pmatrix}^T$

return to the initial position. We can see that the joint positions converge much slower than with friction included in the model.

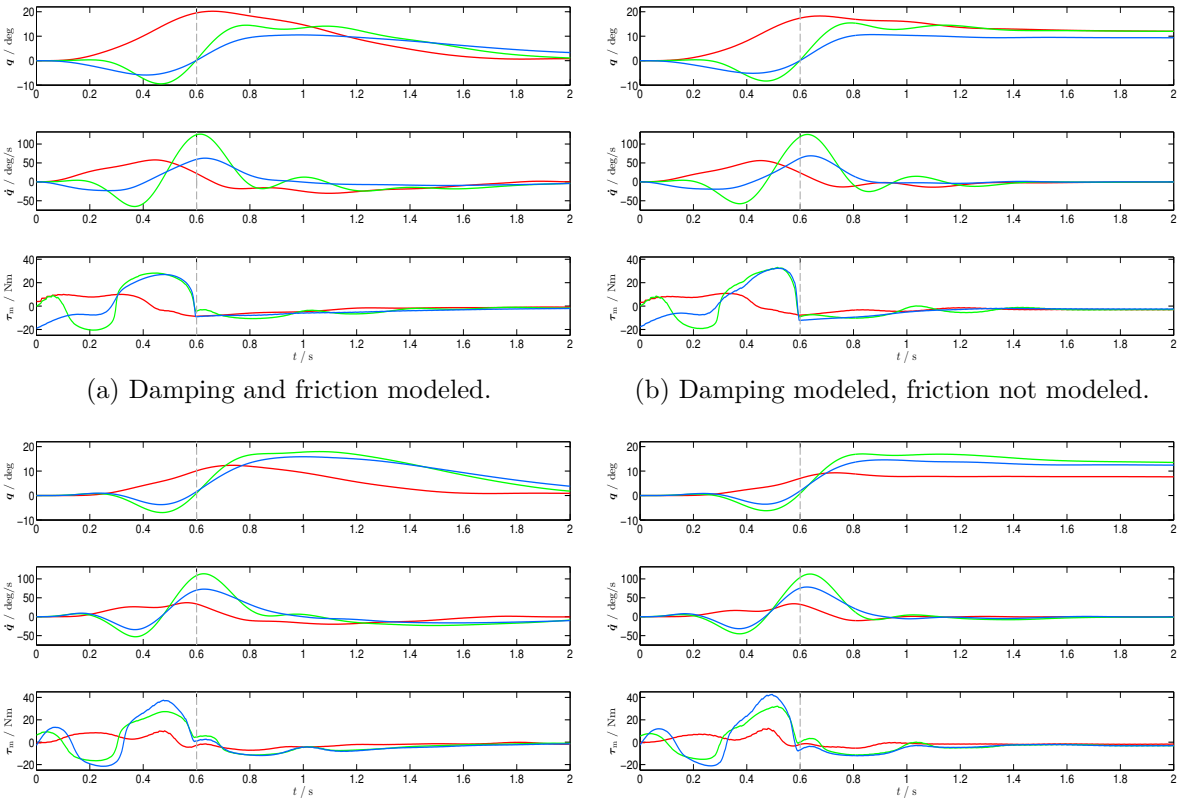
The key message of this subsection is that a difference of the model to the plant leads to deviations in the task and desired goal. Moreover, the worst case is to be unaware of the damping. Whereas a divergence in the friction has less impact.

5.5.3. Flexible bearing included

Until now the plant had the same DOF as the model, i. e. no flexible bearing. Now we change the plant to have the flexible bearing included (see Eq. (3.20)), while the model remains unaware of the flexible bearing. The result is given in Figure 5.6 and Table 5.2.

Again, the desired goal is $\boldsymbol{p}_\mathrm{d} = (\,0.00\ 0.00\ 2.0233\,)^T\,\mathrm{m}$, $\boldsymbol{v}_\mathrm{d} = (\,1.7629\ 1.7629\ 0.00\,)^T\,\mathrm{m\,s^{-1}}$ and $t_\mathrm{d} = 0.6\,\mathrm{s}$. Including the bearing into the plant leads to a deviation in the reached position and velocity which is acceptable if a loss of accuracy is tolerated. For the position the accuracy is in a tolerance which is acceptable. The accuracy of the velocity is worse than the one for the position. But having the task in mind of playing a ball back to the audience, the tolerance on the accuracy can be enlarged.

Let us assume a ball game between two humans. Where one person throws the ball and the other has to hit the ball back with its head. We all can imagine that the ball will not always be returned exactly to the opponents position. Hence, we tolerate this divergence for now.



(a) Damping and friction modeled.

(b) Damping modeled, friction not modeled.

(c) Damping not modeled, friction modeled.

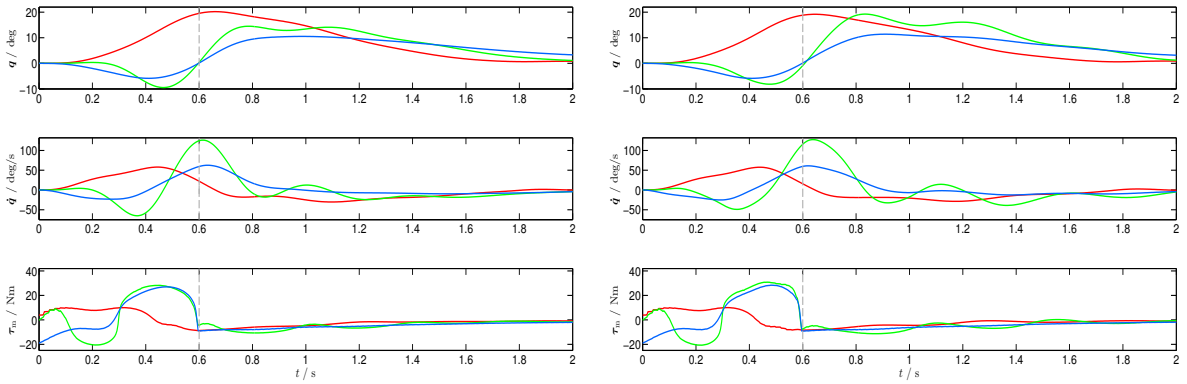
(d) Damping and friction not modeled.

Figure 5.5.: Comparison of model deviations from the simulated plant. The desired position is $\boldsymbol{p}_\mathrm{d} = \begin{pmatrix} 0.00 & 0.00 & 2.0233 \end{pmatrix}^T$ m and the velocity $\boldsymbol{v}_\mathrm{d} = \begin{pmatrix} 1.7629 & 1.7629 & 0.00 \end{pmatrix}^T \mathrm{m\,s^{-1}}$ have to be reached at $t_\mathrm{d} = 0.6\,\mathrm{s}$. Red, green, and blue denote yaw, pitch, and roll axis respectively.

But how can this divergence be explained? Let us have a look into the motions, explicitly to the torque of the pitch joint. This is at around 0.4 s increasing more than without the bearing in the plant. And again we have a problem where the current limitation results in a limitation of the acceleration and so the goal speed can not be met.

5.5.4. Experiments on the Robotic System

In this subsection the experiment differs in the implementation of nonlinear iterations. We had to find a way to test the TLOC principle in an easy way, because the nonlinear iterations would take too much time to implement. The implementation we used in the simulations would not work without a revised version on the microcontroller and PC due to delays in the communication and computation. It was already mentioned in this chapter that we expect the feedback gains to be computed and transferred to the



(a) Plant without the flexible bearing.

(b) Plant simulates the flexible bearing.

Figure 5.6.: Comparison of deviations between plant with and without the flexible bearing simulated. The desired position is $\boldsymbol{p}_\mathrm{d} = \begin{pmatrix} 0.00 & 0.00 & 2.0233 \end{pmatrix}^T \mathrm{m}$ and the velocity $\boldsymbol{v}_\mathrm{d} = \begin{pmatrix} 1.7629 & 1.7629 & 0.00 \end{pmatrix}^T \mathrm{m\,s^{-1}}$ have to be reached at $t_\mathrm{d} = 0.6\,\mathrm{s}$. Red, green, and blue denote yaw, pitch, and roll axis respectively.

microcontroller in zero time for our simulations, i. e. faster than the sample time on the microcontroller (4 ms). It would have taken to much time to implement the nonlinear iterations considering the timing behavior of transmission and computation, this could not be fulfilled in this thesis.

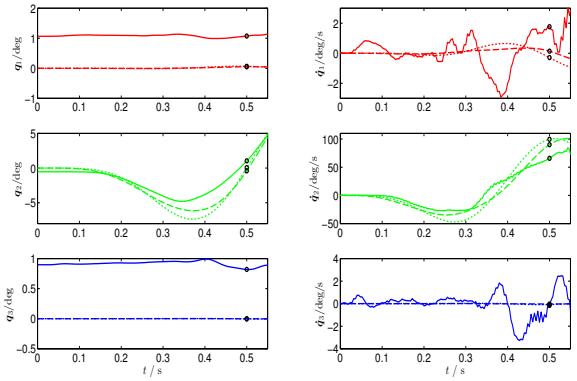
Therefore, we adapt the code given in Pseudo Code 1 to an open loop code 2. Here we just compute the gains by iterating the TLOC four times starting every time from the time t_0 and use the model without the flexible bearing to predict the new states and commands. The feedback gains found are hard coded on the microcontroller for testing. Looking at the code we see a difference in the iterations. Now, we start in every iteration with the initial state $\boldsymbol{x}_0 = \boldsymbol{0}$. The feedback gains coded to the microcontroller are just for the hit motion, afterwards we switch back to position control[3].

For this task we decided to use the easiest motion we could imagine, i. e. batting a ball towards our front (in x-direction). Our goal is to be at the desired time $t_\mathrm{d} = 0.5\,\mathrm{s}$ at the position $\boldsymbol{p}_\mathrm{d} = \begin{pmatrix} 0.00 & 0.00 & 2.0233 \end{pmatrix}^T \mathrm{m}$ with a velocity of $\boldsymbol{v}_\mathrm{d} = \begin{pmatrix} 1.7629 & 0.00 & 0.00 \end{pmatrix}^T \mathrm{m\,s^{-1}}$.

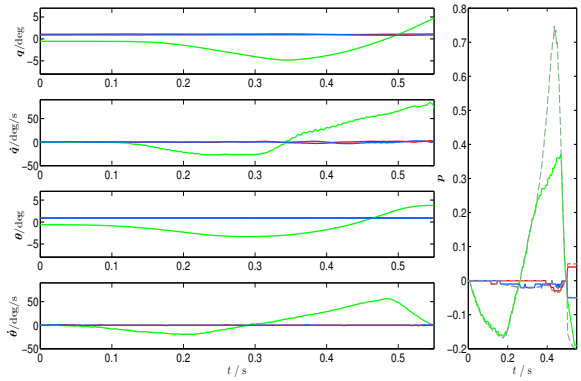
What must be the motion to hit a ball to the front standing upright? Let us assume that we as human should do the task of hitting the ball with our forehead. We would go back, accelerate and hit the ball. This is easy to imagine. Let's see what the robot does (cf. Fig. 5.7b). It starts its motion by swinging back the head (green line). The motor starts and the link follows, as described by the coupling $\boldsymbol{\tau}_\mathrm{c}$. Then it turns direction to accelerates, seen in the pwm signal and the increase of velocity. Basically the motion that we expected.

But let's have a closer look by comparing the robot with our plant simulations (cf. Fig. 5.7a). We see a deviating motion compared to the simulated motion with (dashed)

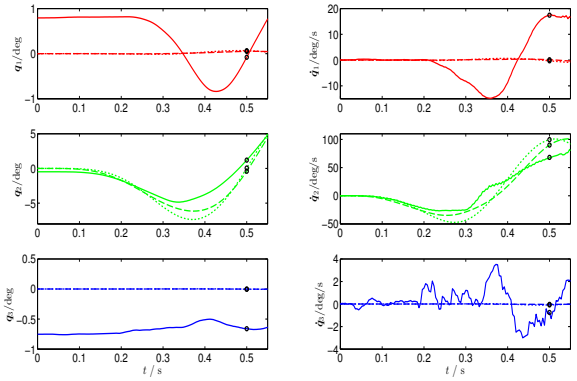
[3]Using either a P-controller or an infinite horizon LQR we implemented within this thesis to test if it is stable and does not oscillate.



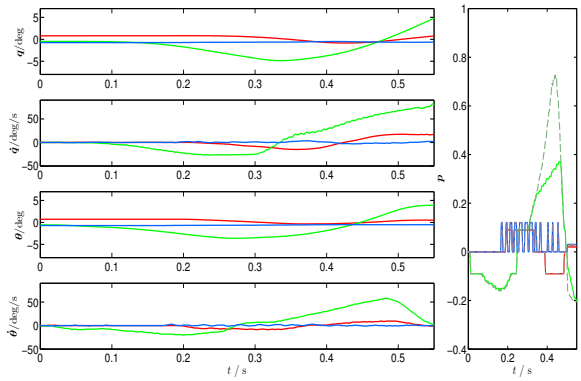
(a) No compensation.



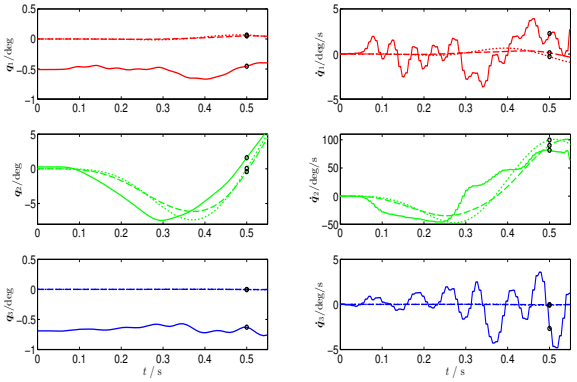
(b) No compensation.



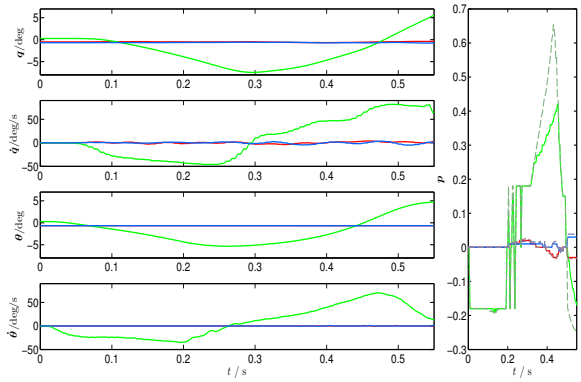
(c) Compensating static friction.



(d) Compensating static friction.



(e) Compensating static and Coulomb friction.



(f) Compensating static and Coulomb friction.

Figure 5.7.: Comparison of simulation (dashed with, dotted without flexible bearing) and robot (solid) behavior. The desired position $\boldsymbol{p}_\mathrm{d} = \begin{pmatrix} 0.00 & 0.00 & 2.0233 \end{pmatrix}^T \mathrm{m}$ and velocity $\boldsymbol{v}_\mathrm{d} = \begin{pmatrix} 1.7629 & 0.00 & 0.00 \end{pmatrix}^T \mathrm{m\,s^{-1}}$ have to be reached at $t_\mathrm{d} = 0.5\,\mathrm{s}$ (marked). Red, green, and blue denote yaw, pitch, and roll axis respectively. On the right the robot states with pwm. The controller sets the pwm (dashed), which is current limited (solid).

Pseudo Code 2: Nonlinear iteration description in the view of the PC.

set $n = 0$
set initial state $\boldsymbol{x}_0 = \mathbf{0}$
Set $\bar{\boldsymbol{K}}_{n\ldots N-1} = \mathbf{0}$
initialize $\bar{\boldsymbol{A}}_{n\ldots N}, \bar{\boldsymbol{B}}_{n\ldots N}$ around linearization points $\boldsymbol{x}_0$ and $\boldsymbol{u} = \mathbf{0}$
while $k = 0$ *to* 3 **do**
    reset $n = 0$
    predict system behavior $\bar{\boldsymbol{A}}_{n\ldots N},\ \bar{\boldsymbol{B}}_{n\ldots N},\ \bar{\boldsymbol{x}}_{p,n\ldots N},\ \boldsymbol{u}_{p,n\ldots N-1}$
    quadratize cost $\bar{\boldsymbol{Q}}_{n\ldots N},\ \bar{\boldsymbol{R}}_{n\ldots N-1},\ \bar{\boldsymbol{S}}_{n\ldots N-1}$ for $\bar{\boldsymbol{x}}_{p,n\ldots N},\ \boldsymbol{u}_{p,n\ldots N-1}$
    compute gains $\bar{\boldsymbol{K}}_{n\ldots N-1}$
end
compute $N_\mathrm{d} = \left\lfloor \frac{t_\mathrm{d}}{T_\mathrm{s}} \right\rfloor$
set gain matrices $\bar{\boldsymbol{K}}_{n\ldots N_\mathrm{d}}$ to microcontroller

and without (dotted) the flexible bearing. The principle motion is the same: moving backwards and accelerate to the front. But the amount of velocity which is met by the simulations is not reached. Note that the simulations both are only minimal deviating from the desired goal (cf. Table 5.3).

One possible reason might be the static friction. In the calibration we assumed to have no static friction and also neglect this in the model, as our aim is to control the robot in motion, where no static friction is active. Therefore, we checked if by compensating the static friction, the robot behaves better (cf. Fig. 5.7c and 5.7d). We see no big difference to the one before. The velocity is getting slightly better. The motion itself keeps its behavior as desired.

Another reason might be an inaccurate calibration of the Coulomb friction. Thus, we adapted the program in a manner, that Coulomb and static friction are compensated on the pitch axis – because this is the axis of interest in this example. The Coulomb friction is linearly compensated in dependency of the motors velocity. Low velocity coincides with a high compensation, high velocity leads to a lower compensation. The result shows Figure 5.7e and 5.7f. Now we get closer to the desired velocity. Thus, we could say that the friction might be modeled inaccurate. The friction is maybe not a sigmoid function as expected and the static friction should better be included in the model.

I also want to add another reason that could cause the divergence. In Figures 5.7b, 5.7d and 5.7f have a look at the pwm which should be set (dashed) and the one which is set (solid) after current limitation. Both lines coincide until the point where the controller wants to accelerate to reach the desired velocity. The controller knows the limitations by the limiting torque computed with Equation (5.7). However, the gains compute a pwm higher than the limit. One reason therefor is that the gains are computed based on a linearization/quadratization at the predicted states. If the actual states are far away from that, this is not valid anymore. We saw in the previous subsections that if the

Table 5.3.: Comparison between robot and simulation accuracy. The desired position is $\boldsymbol{p}_\mathrm{d} = (\,0.00\;0.00\;2.0233\,)^T\,\mathrm{m}$ and the velocity $\boldsymbol{v}_\mathrm{d} = (\,1.7629\;0.00\;0.00\,)^T\,\mathrm{m\,s^{-1}}$ have to be reached at $t_\mathrm{d} = 0.5\,\mathrm{s}$.

	robot	robot static friction compensated	robot static and Coulomb friction compensated	simulation plant with bearing	simulation plant without bearing
$\boldsymbol{p}(t_\mathrm{d})$ / m	$\begin{pmatrix}0.0167\\0.0131\\2.0230\end{pmatrix}$	$\begin{pmatrix}0.0180\\-0.0130\\2.0230\end{pmatrix}$	$\begin{pmatrix}0.0235\\-0.0097\\2.0229\end{pmatrix}$	$\begin{pmatrix}-0.0254\\0.0000\\2.0230\end{pmatrix}$	$\begin{pmatrix}0.0010\\-0.0001\\2.0233\end{pmatrix}$
$\boldsymbol{v}(t_\mathrm{d})$ / $\mathrm{m\,s^{-1}}$	$\begin{pmatrix}1.2708\\0.0832\\-0.0219\end{pmatrix}$	$\begin{pmatrix}1.3080\\0.0316\\-0.0233\end{pmatrix}$	$\begin{pmatrix}1.3902\\-0.0270\\-0.0341\end{pmatrix}$	$\begin{pmatrix}1.7539\\0.0004\\0.0379\end{pmatrix}$	$\begin{pmatrix}1.7546\\0.0006\\-0.0017\end{pmatrix}$

model deviates from the plant, the deviations in the desired goal are increasing. At this point we have to review and say that our model from the previous chapter might not be accurate enough to control the robot precisely. But we can show that the principle of TLOC works on the robot – the motion is just the one we expected to be and coincides with the simulations overall behavior.

This is much better to visualize in a motion video[4] than in figures. This shows simulation and real behavior side by side. The motions are only shown from the starting to the desired time in slow motion. Here, we can see that the intention of the motion is realized on the robot. However, the precision of the model is unequaled in the desired goal.

5.6. Summary

This chapter briefly described the TLOC principle as a summary of the papers [Schüthe et al. 2014, 2015]. The controller is operating in a linear (microcontroller) and a nonlinear (PC) stage to fulfill the task of being at position $\boldsymbol{p}_\mathrm{d}$ with velocity $\boldsymbol{v}_\mathrm{d}$ at the desired time t_d. The contribution made is the development of the Task Level Optimal Control which computes feedback gains for the linear stage to run a fast control for the task of hitting balls. The nonlinear stage is called on every camera update to improve the motion and recompute new gains that fit to the new criteria given by the current state. This implementation has been tested in a simulation run.

[4] `http://www.informatik.uni-bremen.de/agebv2/downloads/videos/schuetheThesisMotionExamples.mp4`

We have shown the impact of model deviations and the difference of the control having a model neglecting the flexible bearing, but using a plant with and without the flexible bearing.

Finally, we showed a reduced experiment on the real robot that, for implementation reasons, worked without the nonlinear iterations. So there is just the linear acting feedback controller with optimal gains computed in the beginning. It could be shown that the intention of motion is met, but the accuracy in desired position and velocity insufficiently. This has mainly to do with an improper modeling of the friction. Moreover, it must be considered that there are other effects that are not modeled, e. g. a play in the coupling spring. But it could be shown that the overall principle of the TLOC works.

6
Human-Robot Interaction

Let us now take a look at the human-robot interaction (HRI) of "Doggy". We are going to do this in two sections. The first describes the game tested and showed on the Open Campus day of the University of Bremen in July 2015. We briefly discuss the setting and the overall audience reaction. The second illustrates an implementation of acoustic sound recognition to interact with the audience, which was a master thesis supervised by me.

6.1. Ball Playing Robot

The entertainment aspect has been demonstrated on the Open Campus day at the University of Bremen in 2015 (Fig. 6.1). We presented Doggy for the first time having its new interior, i. e. the newest version described in Chapter 2. As we finished the robot only a few weeks before the open campus day, a P-controller was implemented on the microcontroller to regulate the motor positions. The communication to the PC was handled by USB. To make the robot play the ball game its predecessor "Piggy" did [Laue et al. 2013], we had to adapt the PC software and calibrate the stereo camera system[1], both with the existing code of Piggy.

The setting was chosen in an outdoor environment at the campus. The cameras had to deal with changing light conditions and with an audience of all ages. The nice thing with our entertainment robot is that it is self-explanatory. If one person throws a ball others get the game's intention immediately. Children are enthusiastic to play with Doggy, perhaps caused by its "fluffy" costume. The costume helps to attract both technophile and non-technophile people. Thus, the demonstrator addressed all sorts of people.

[1] Both parts were fulfilled within this thesis.

Figure 6.1.: Doggy at the Open Campus day 2015.

On the campus the robot was placed next to a sidewalk. A crowd was building around it for the most time. First, the new visitors were just looking. But by passing them a ball they get involved very fast and easily. For those who want to have more information about the behavior and the implementation of the game, three members of our group were happy to answer their questions and showed them the ball tracker on a screen in real-time.

For the game we recognized a great amount of ball hits, when the trajectory was towards the intersection of the robot's workspace. Also children from the age 5 could take part of the game and managed to throw the ball precise enough to make Doggy hit it. Children under that age often did not have the power to throw the ball that high and far (at least 3 m, best is between 6 m and 10 m).

Problems we had on that day, were recognition problems and connection problems. The USB connection broke several times. A reset handled this problem. Another problem were trees, because their leaves were sometimes recognized as valid trajectories, leading to a motion of the robot. In the meantime we had to fix the head, as one person threw to fast and destroyed the Styrofoam head. We fixed that with tape. Therefore, we replaced the broken sphere by a new one that was entirely fixed with aluminum tape inside and outside in a honeycomb structure afterwards.

Overall, we could say that this was a successful test of the hard- and software and showed that the interior of plexiglass can hold the forces and impacts that happened during the game. The robot ran for 7 hours. A summarizing video is shown on the working groups website[2].

But, it showed the need of hitting the ball precisely. Most balls were hit back randomly and delayed the game due to getting the balls back. A play back of the balls would simplify the setting a lot.

[2] `http://www.informatik.uni-bremen.de/agebv2/downloads/videos/doggyOpenCampusDay.mp4`

6.2. Acoustic Orientation

Playing ball games is the main task of our entertainment robot, therefore it has to be agile. But for entertainment, it needs more than just the capability to hit balls back to the audience. The robot must interact with the audience to be an accepted opponent. One interaction method is a human-like reaction to sound, i. e. turning towards the sound source.

In [Bartsch 2014] we implemented a sound localization using stereo microphones on Doggy to react to sound like a human would do, i. e. by turning its view to the sound source. The localization is based on a frequency learning filter – to limiting the amplitudes of frequencies which are heard always – and adaptation to the environmental noise level. This is important as the robot's purpose is to provide entertainment at arbitrary places with groups of people and noisy things happening in the environment. The approaches are inspired by human behavior. The implementation was tested under laboratory conditions with specified sounds as well as with an audience to show that the localization helps a robot to interact with humans.

The direction of the sound source in two-dimensional space can be estimated using a stereo microphone. A standard algorithm for this is triangulation [Sasaki et al. 2006], but only if the source is in the far-field, i. e. the distance to the sound source is greater than the distance d between the microphones – for angle errors less than 5°, the source should be at least ten times the distance d. This approximation works for us, as the distance to throw a ball is at least 3 m. Using the time delay Δt between the microphones, we are able to compute the angle between the robot's x axis and the source, using the speed of sound c.

$$\alpha = \frac{\pi}{2} - \arccos\left(\frac{\Delta t c}{d}\right) \tag{6.1}$$

We use a cross correlation function (CCF) to detect the time delay between left and right microphone. For more details see [Bartsch 2014].

The implementation of the sound localization is done on the PC. After a sound is detected the robot turns towards the estimated angle using the first joint (yaw).

Doggy's new feature was tested in two different student classes (cf. Fig. 6.2). The students were only told to attract the robot's attention by making sounds. They tested the sounds which were interesting in their mind. The result was filmed from two perspectives, towards the robot and from the robot's view[3]. During these experiments, Doggy regularly turned to the sound source, iff the sound was louder than the environmental noise. As reference we used our impression, if we would have been turned to that sound. In most cases Doggy's motion and our impression were equal. However, speech was a problem for our detector, in many cases, the robot turned only if a hard vowel was in the word. The students have been enthusiastic about the robot's reaction and were also astonished during the first move. And when the first sound was detectable for Doggy

[3] `http://www.informatik.uni-bremen.de/agebv2/downloads/videos/schuetheDoggyNoiseLocalization.mp4`

Figure 6.2.: Doggy standing in front of his uninformed audience.

and it turns toward that source, we had the attention of the whole class which was the aim of this work.

7 Conclusion

We started with the question: "Is it possible to define a rebound task for our robot within a controller?". We can now answer this question with yes. We showed in Chapter 5 that it is possible in simulation and on the robot. The Task Level Optimal Control (TLOC) principle works, which uses an optimal controller (LQR), the task is put into cost functions to give the controller the freedom of how to fulfill this task. The batting motion has been seen on the robot and compared to the simulation, which led to a good result. The overall intention of the motion is also met on the robot using the feedback gains for the whole task precomputed by simulation iterations. It could be shown that this implementation decides the best way of fulfilling the task – we could say optimally decides based on the optimality criterion of the controller. Moreover, the soft limitations we implemented as cost functions lead the controller to know the physical system and thus it was able to actively use redundancy to fulfill a task. This is nice and elegant. Torque limits could be met using this strategy, too.

A challenge that this controller brings with it, is the determination of parameters to describe the task, mostly these are weighting parameters. Finding a set of parameters that on one hand fulfills the task with highest accuracy, but on the other hand should fulfill the soft constraints, is very difficult – or impossible. Finding those parameters might be handled by an optimization problem, where the goal is to minimize the difference between desired goal and achieved goal. Moreover, we could say that the task should be fulfilled with an allowed accuracy and the optimization should find the parameters for it.

However, if we think of our rebound task, the accuracy needed to fulfill the task needs to be tested in the future. Moreover, it might be more accurate after implementation of the nonlinear iteration process. This includes following steps: (1.) Reading the actual state and time of the microcontroller. (2.) Based on that time we have to predict the states and commands. (3.) Knowing the computation time (which is deterministic) for

the feedback gains and the delay time for transmitting the gains to the microcontroller, we must transmit only the most current gains which become active at the microcontroller, i. e. do not transmit feedback gains from the past.

Another idea I had was the redefinition of the task. The task should not be to reach a desired position with the robot, but to reach a desired position with the ball that was played. Therefore, the state has to be enhanced by the position and velocity of the ball, i. e. the ball trajectory dynamics known from physics. This leads to a new state space with two decoupled dynamics – the robot and ball dynamics. Generally, the dynamics are decoupled, except for the batting, where both dynamics influence each other. This is the point where the robot has the ability to change the ball's trajectory to meet the desired goal. The desired goal position of the ball would be implemented as cost function that needs to be minimized. And by defining the impact the head has on the ball within the costs. This implementation would be more elegant as a global task definition, because the computation of where to hit the ball and how is put into the controller and solved in an optimal way. However, one needs to take care about those balls that can not be hit at all (not intersection with the robot's workspace). It might be able to put this into a cost function, too.

In our calibration (Chapter 4) we found two models. One describing the behavior of the robot without the flexible bearing of the yaw axis and one with the flexible bearing. Both models were using gyroscopes as link velocity and position estimator which is also a contribution of our work. We showed that this works by implementing a Kalman filter on the microcontroller using the model with flexible bearing of the yaw axis included. The states found fit the measured data well. Especially, we showed a combination of sensors to estimate link and motor positions and velocities, that has rarely been used and investigated before. But it turned out that this combination has a high accuracy of correctly estimate the state. However, the calibrated model we used in the prediction is not precise enough and should be improved. *But how can the model be further improved to fit better than both of the models identified?*

One idea that comes into mind is to split the calibration into several subcalibrations. E. g. by calibrating the motor parameters first, without knowing the system at all. Another idea is to include the stereo camera into this calibration process. If the stereo camera system is already calibrated, we can accurately measure Cartesian distances of specific points in the left and right image. If we put a marker on Doggy's head in such a way that the camera is able to see this marker, we could measure the link position using the cameras and compare this to the estimated link position given by the gyroscopes. This could improve the calibration.

Another idea is to redefine the models given in Chapter 3 which describe the system denoted in Chapter 2. During the calibration procedure and throughout experiments we found some behaviors that were not included in our models, i. e. the dynamics. Some of them have been fixed already before the calibration, e. g. that screws were loosened by the vibrations leading to a play when the motor changes direction. Or that the tooth wheels were not tightened fast enough on the shaft, which leads to the same phenomenon. But there are some things we have not changed and changing these might improve the

robot. First of all the bearing of the yaw axis should be replaced by a more stiff bearing to overcome the flexibility at that point. Which would lead to better measurements. Another point of improvement is given in the tooth belt of the pitch axis. Here, only one wheel is used to tighten the tooth belt. Thus, the tooth belt could not be tightened fast enough and leads to a play. Additionally, there could be elasticity in the links. But, to implement this into the dynamics would rapidly increasing the state space, which could hardly be handled. Especially if the system should operate in real time.

Finally, I want to summarize this work again. We have contributed a TLOC for a ball batting task on an entertainment robot [Schüthe et al. 2014, 2015]. The controller runs in two stages, one at a frequency of 250 Hz on a microcontroller (linear stage) and the other at 50 Hz on the PC as a nonlinear stage. Another contribution was made by the calibration procedure where we use only the encoder data of the motors and the gyroscopes of the Inertial Measurement Units (IMUs) to estimate the link velocities and positions [Schüthe et al. 2016]. We enhanced this work by a flexible bearing model and a Kalman filter that estimates the robot's states in real time. Based on these models and the states estimated on the microcontroller, we were able to implement a ball hitting motion using our TLOC. We showed that this model based controller is able to handle the task of hitting balls (in principle). This includes a model that is well reflecting the robotic system.

In my opinion we have successfully build an entertainment robot that can withstand the forces acting during the bat motion, interacts with the audience – and I hope some more interactions, like people recognition with cameras, will follow; and is able to play a ball game which is a demanding task. I think finding the most accurate model takes a lot of time, since the acrylic glass construction caused unwanted problems. We found a model that handles most phenomena – not all – and is precise enough to build a Linear Quadratic Regulator (LQR) and our TLOC algorithm on it. Furthermore, the TLOC fulfills the desired task on the robot and it looks very dynamic and sporty, more like a human than a robot. We are more than)satisfied with how the robot's motion turned out.

Publicated Work by the Author

Schüthe, Dennis (2015). "Dynamic Rebound Control and Human Robot Interaction of a Ball Playing Robot". In: *Formal Modeling and Verification of Cyber-Physical Systems*. Ed. by Rolf Drechsler and Ulrich Kühne. Vol. 1. 1st International Summer School on Methods and Tools for the Design of Digital Systems, Bremen, Germ. Not peer-reviewed

My share is 100%.

Give an overall discription of the work. Springer Vieweg, pp. 299–301.

Schüthe, Dennis and Frese, Udo (2014). "Task Level Optimal Control of a Simulated Ball Batting Robot". In: *ICINCO 2014 – 11th International Conference on Informatics in Control, Automation and Robotics*. Ed. by Joaquim Filipe et al. Vol. 2.

My share is 90%.

I implemented the proposed simulation of the Task Level Optimal Controller and did the experiments. This work was presented by me in Vienna, Austria. SCITEPRESS, pp. 45–56. DOI: 10.5220/0005026100450056.

– (2015). "Optimal Control with State and Command Limits for a Simulated Ball Batting Task". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

My share is 90%.

I implemented the proposed simulation of the Task Level Optimal Controller, added the limitations of torque and joint angles to the TLOC. Implemented the simulation of ball hitting tasks. This work was presented by me in Hamburg, Germany., pp. 3988–3994. DOI: 10.1109/IROS.2015.7353939.

Schüthe, Dennis, Wenk, Felix, and Frese, Udo (2016). "Dynamics Calibration of a Redundant Flexible Joint Robot based on Gyroscopes and Encoders". In: *13th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2016)*. Ed. by Oleg Gusikhin, Dimitri Peaucelle, and Kurosh Madani. Vol. 1.

My share is 70%.

The calibration program has been devolped by me using the toolkit SLOM. The calibration was also done by me. The debugging has been done together with Felix Wenk. The work was presented by me in Lisbon, Portugal. SCITEPRESS, pp. 335–346. DOI: 10.5220/0005976603350346.

References

Aboaf, E. W., Atkeson, C. G., and Reinkensmeyer, D. J. (Apr. 1988). "Task-level robot learning". In: *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, 1309–1310 vol.2. DOI: 10.1109/ROBOT.1988.12245.

Aboaf, E. W., Drucker, S. M., and Atkeson, C. G. (May 1989). "Task-level robot learning: juggling a tennis ball more accurately". In: *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, 1290–1295 vol.3. DOI: 10.1109/ROBOT.1989.100158.

Aertbeliën, E. and Schutter, J. De (Sept. 2014). "eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1540–1546. DOI: 10.1109/IROS.2014.6942760.

Anderson, Brian DO and Moore, John B (2007). *Optimal Control: Linear Quadratic Methods*. Courier Corporation.

Andersson, R. L. (1986). "Living in a Dynamic World". In: *Proceedings of 1986 ACM Fall Joint Computer Conference*. ACM '86. Dallas, Texas, USA: IEEE Computer Society Press, pp. 97–104.

– (Feb. 1989). "Aggressive trajectory generator for a robot ping-pong player". In: *IEEE Control Systems Magazine* 9.2, pp. 15–21. DOI: 10.1109/37.16766.

Argall, Brenna et al. (2006). "The First Segway Soccer Experience: Towards Peer-to-peer Human-robot Teams". In: *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-robot Interaction*. HRI '06. Salt Lake City, Utah, USA: ACM, pp. 321–322. DOI: 10.1145/1121241.1121296.

Bartsch, Michel (2014). „Sound of Interest. Ein Ballspielroboter hört stereo". Magisterarb. University Bremen.

Bäuml, B., Wimböck, T., and Hirzinger, G. (Oct. 2010). "Kinematically optimal catching a flying ball with a hand-arm-system". In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 2592–2599. DOI: 10.1109/IROS.2010.5651175.

Bäuml, B. et al. (May 2011). "Catching flying balls and preparing coffee: Humanoid Rollin'Justin performs dynamic and sensitive tasks". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 3443–3444. DOI: 10.1109/ICRA.2011.5980073.

Behnke, Sven, Müller, Jürgen, and Schreiber, Michael (2006). "RoboCup 2005: Robot Soccer World Cup IX". In: ed. by Ansgar Bredenfeld et al. Berlin, Heidelberg: Springer

Berlin Heidelberg. Chap. Playing Soccer with RoboSapien, pp. 36–48. DOI: 10.1007/11780519_4.

Birbach, O. (2012). "Tracking and Calibration for a Ball Catching Humanoid Robot". doctoral thesis. Universität Bremen; www.uni-bremen.de.

Chen, W. and Tomizuka, M. (Apr. 2014). "Direct Joint Space State Estimation in Robots With Multiple Elastic Joints". In: *IEEE/ASME Transactions on Mechatronics* 19.2, pp. 697–706. DOI: 10.1109/TMECH.2013.2255308.

Cheng, P. and Oelmann, B. (Feb. 2010). "Joint-Angle Measurement Using Accelerometers and Gyroscopes — A Survey". In: *IEEE Transactions on Instrumentation and Measurement* 59.2, pp. 404–414. DOI: 10.1109/TIM.2009.2024367.

Compressorhead (Aug. 23, 2016). URL: https://compressorhead.rocks/ (visited on 08/23/2016).

Craig, John J. (2005). *Introduction to robotics: mechanics and control*. 3. ed., international ed. Pearson education international. Upper Saddle River, NJ [u.a.]: Pearson, Prentice Hall.

De Luca, Alessandro and Book, Wayne (2008). "Robots with Flexible Elements". In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 287–319. DOI: 10.1007/978-3-540-30301-5_14.

De Schutter, Joris et al. (2007). "Constraint-based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty". In: *The International Journal of Robotics Research* 26.5, pp. 433–455. DOI: 10.1177/0278364907078091O7.

Decre, W. et al. (May 2009). "Extending iTaSC to support inequality constraints and non-instantaneous task specification". In: *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pp. 964–971. DOI: 10.1109/ROBOT.2009.5152477.

Deguchi, K., Sakurai, H., and Ushida, S. (Sept. 2008). "A goal oriented just-in-time visual servoing for ball catching robot arm". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3034–3039. DOI: 10.1109/IROS.2008.4650615.

Erdmann, Wlodzimierz S (2013). "Problems of sport biomechanics and robotics". In: *International Journal of Advanced Robotic Systems* 10.

Featherstone, Roy and Orin, David E. (2008). "Dynamics". In: ed. by Bruno Siciliano and Oussama Khatib. Springer Berlin Heidelberg, pp. 35–65.

Geppert, Linda (2004). "Qrio, the robot that could". In: *Ieee Spectrum* 41.5, pp. 34–37.

Goebel, R. (May 2005). "Stabilizing a linear system with Saturation Through optimal control". In: *IEEE Transactions on Automatic Control* 50.5, pp. 650–655.

Goretkin, G. et al. (May 2013). "Optimal sampling-based planning for linear-quadratic kinodynamic systems". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 2429–2436. DOI: 10.1109/ICRA.2013.6630907.

Grotjahn, M., Daemi, M., and Heimann, B. (2001). "Friction and rigid body identification of robot dynamics". In: *International Journal of Solids and Structures* 38.10–13, pp. 1889–1902. DOI: http://dx.doi.org/10.1016/S0020-7683(00)00141-4.

Hammer, Tobias (Sep. 2011). „Aufbau, Ansteuerung und Simulation eines interaktiven Ballspielroboters“. Magisterarb. Universität Bremen.

Hegyi, Andreas, De Schutter, Bart, and Hellendoorn, Hans (June 2005). “Model predictive control for optimal coordination of ramp metering and variable speed limits”. en. In: *Transportation Research Part C: Emerging Technologies* 13.3, pp. 185–209.

Hertzberg, Christoph, Wagner, René, and Frese, Udo (2012). “Tutorial on Quick and Easy Model Fitting Using the SLoM Framework”. In: *Spatial Cognition VIII*. Ed. by Cyrill Stachniss, Kerstin Schill, and David Uttal. Vol. 7463. Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, pp. 128–142.

Hoshiya, Masaru and Saito, Etsuro (1984). “Structural Identification by Extended Kalman Filter”. In: *Journal of Engineering Mechanics* 110.12, pp. 1757–1770. DOI: 10.1061/(ASCE)0733-9399(1984)110:12(1757).

Hu, Jwu-Sheng et al. (2010). “A ball-throwing robot with visual feedback”. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 2511–2512. DOI: 10.1109/IROS.2010.5649335.

Ishida, T. (Oct. 2004). “Development of a small biped entertainment robot QRIO”. In: *Micro-Nanomechatronics and Human Science, 2004 and The Fourth Symposium Micro-Nanomechatronics for Information-Based Society, 2004. Proceedings of the 2004 International Symposium on*, pp. 23–28. DOI: 10.1109/MHS.2004.1421265.

Jeon, Soo, Tomizuka, Masayoshi, and Katou, Tetsuaki (2009). “Kinematic Kalman filter (KKF) for robot end-effector sensing”. In: *Journal of dynamic systems, measurement, and control* 131.2, p. 021010.

Johansen, Tor A., Petersen, Idar, and Slupphaug, Olav (2002). “Explicit sub-optimal linear quadratic regulation with state and input constraints”. In: *Automatica* 38.7, pp. 1099–1111.

Kalman, Rudolph Emil (1960). “A new approach to linear filtering and prediction problems”. In: *Journal of basic Engineering* 82.1, pp. 35–45.

KG, 4attention GmbH & Co. (Aug. 23, 2016). *Robokeeper website*. URL: http://robokeeper.com/ (visited on 08/23/2016).

Kober, J., Glisson, M., and Mistry, M. (Nov. 2012a). “Playing catch and juggling with a humanoid robot”. In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pp. 875–881. DOI: 10.1109/HUMANOIDS.2012.6651623.

– (Nov. 2012b). *Playing catch and juggling with a humanoid robot*. Ed. by DisneyResearchHub. URL: https://youtu.be/83eGcht7IiI (visited on 08/23/2016).

KUKA (May 2016). *KUKA robots won the hearts of Eurovision audiences*. URL: http://www.kuka-robotics.com/en/pressevents/news/NN_20160525_KUKA_Eurovisio.htm (visited on 08/23/2016).

Kurze, Matthias et al. (2008). “Modellbasierte Regelung von Robotern mit elastischen Gelenken ohne abtriebsseitige Sensorik”. PhD thesis. Technische Universität München.

Laue, Tim et al. (2013). “An Entertainment Robot for Playing Interactive Ball Games”. In: *RoboCup 2013: Robot Soccer World Cup XVII*. Lecture Notes in Artificial Intelligence. to appear. Springer.

Lenz, C. et al. (Oct. 2009). "Constraint task-based control in industrial settings". In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3058–3063. DOI: 10.1109/IROS.2009.5354631.

Li, W. and Todorov, E. (2007). "Iterative linearization methods for approximately optimal control and estimation of non-linear stochastic system". In: *International Journal of Control* 80.9, pp. 1439–1453. DOI: 10.1080/00207170701364913.

Li, Weiwei and Todorov, Emanuel (2004). "Iterative linear quadratic regulator design for nonlinear biological movement systems." In: *ICINCO (1)*, pp. 222–229.

Lofaro, D. M., Sun, C., and Oh, P. (Nov. 2012). "Humanoid pitching at a Major League Baseball game: Challenges, approach, implementation and lessons learned". In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pp. 423–428. DOI: 10.1109/HUMANOIDS.2012.6651554.

Luca, A. De, Schroder, D., and Thummel, M. (Apr. 2007). "An Acceleration-based State Observer for Robot Manipulators with Elastic Joints". In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 3817–3823. DOI: 10.1109/ROBOT.2007.364064.

Mare, José B. and Doná, José A. De (2007). "Solution of the input-constrained LQR problem using dynamic programming". In: *Systems & Control Letters* 56.5, pp. 342–348.

Matsushima, M. et al. (Aug. 2005). "A Learning Approach to Robotic Table Tennis". In: *IEEE Transactions on Robotics* 21.4, pp. 767–771. DOI: 10.1109/TRO.2005.844689.

Moberg, Stig (2010). "Modeling and Control of Flexible Manipulators". PhD thesis. Linköping University, Automatic Control, p. 101.

Mülling, Katharina et al. (2013). "Learning to select and generalize striking movements in robot table tennis". In: *The International Journal of Robotics Research* 32.3, pp. 263–279.

Nakai, H. et al. (1998). "A volleyball playing robot". In: *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*. Vol. 2, 1083–1089 vol.2. DOI: 10.1109/ROBOT.1998.677234.

Narendra, K. S. and Parthasarathy, K. (Mar. 1990). "Identification and control of dynamical systems using neural networks". In: *IEEE Transactions on Neural Networks* 1.1, pp. 4–27. DOI: 10.1109/72.80202.

Olsson, H. et al. (1998). "Friction Models and Friction Compensation". In: *European Journal of Control* 4.3, pp. 176–195. DOI: http://dx.doi.org/10.1016/S0947-3580(98)70113-X.

Perez, A. et al. (May 2012). "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics". In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 2537–2542. DOI: 10.1109/ICRA.2012.6225177.

Pham, M. T., Gautier, M., and Poignet, P. (2001). "Identification of joint stiffness with bandpass filtering". In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. Vol. 3, 2867–2872 vol.3. DOI: 10.1109/ROBOT.2001.933056.

Pransky, Joanne (2001). "AIBO – the No. 1 selling service robot". In: *Industrial Robot: An International Journal* 28.1, pp. 24–26. DOI: 10.1108/01439910110380406.

Press, William H et al. (2002). *Numerical Recipes in C++: The Art of Scientific Computing (2nd edn)*. Cambridge UP.

Quigley, M. et al. (Oct. 2010). "Low-cost accelerometers for robotic manipulator perception". In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 6168–6174. DOI: 10.1109/IROS.2010.5649804.

Reist, P. and Tedrake, R. (2010). "Simulation-based LQR-trees with input and state constraints". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 5504–5510. DOI: 10.1109/ROBOT.2010.5509893.

Riley, Marcia and Atkeson, Christopher G. (2002). "Robot Catching: Towards Engaging Human-Humanoid Interaction". In: *Autonomous Robots* 12.1, pp. 119–128. DOI: 10.1023/A:1013223328496.

Sasaki, Yoko, Kagami, Satoshi, and Mizoguchi, Hiroshi (2006). "Multiple sound source mapping for a mobile robot by self-motion triangulation". In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, pp. 380–385.

Schraft, Rolf Dieter et al. (2001). "A mobile robot platform for assistance and entertainment". In: *Industrial Robot: An International Journal* 28.1, pp. 29–35. DOI: 10.1108/01439910110380424.

Scokaert, P.O.M. and Rawlings, J.B. (Aug. 1998). "Constrained linear quadratic regulation". In: *Automatic Control, IEEE Transactions on* 43.8, pp. 1163–1169.

Senoo, T., Namiki, A., and Ishikawa, M. (2006). "Ball control in high-speed batting motion using hybrid trajectory generator". In: *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 1762–1767. DOI: 10.1109/ROBOT.2006.1641961.

Serra, Diana et al. (2016). "An Optimal Trajectory Planner for a Robotic Batting Task: The Table Tennis Example". In: *13th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2016)*. Ed. by Oleg Gusikhin, Dimitri Peaucelle, and Kurosh Madani. Vol. 2. SCITEPRESS, pp. 90–101.

Silva, R., Melo, F. S., and Veloso, M. (Sept. 2015). "Towards table tennis with a quadrotor autonomous learning robot and onboard vision". In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 649–655. DOI: 10.1109/IROS.2015.7353441.

Slotine, Jean-Jacques E. and Li, Weiping (1987). "On the Adaptive Control of Robot Manipulators". In: *The International Journal of Robotics Research* 6.3, pp. 49–59. DOI: 10.1177/027836498700600303.

Somani, N. et al. (Sept. 2015). "Constraint-based task programming with CAD semantics: From intuitive specification to real-time control". In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 2854–2859. DOI: 10.1109/IROS.2015.7353770.

Staufer, Peter and Gattringer, Hubert (2012). "State estimation on flexible robots using accelerometers and angular rate sensors". In: *Mechatronics* 22.8, pp. 1043–1049. DOI: http://dx.doi.org/10.1016/j.mechatronics.2012.08.009.

Tilden, Mark W (2004). "Neuromorphic robot humanoid to step into the market". In: *The Neuromorphic Engineer* 1.1, p. 12.

Vukosavic, Slobodan N. (2013). *Electrical machines*. Power Electronics and Power Systems. New York, NY [u.a.]: Springer.

Vuong, Ngoc Dung and Ang Jr, Marcelo H (2009). "Dynamic model identification for industrial robots". In: *Acta Polytechnica Hungarica* 6.5, pp. 51–68.

Waldron, Kenneth and Schmiedeler, James (2008). "Kinematics". In: ed. by Bruno Siciliano and Oussama Khatib. Springer Berlin Heidelberg, pp. 9–33.

Wan, E. A. and Merwe, R. Van Der (2000). "The unscented Kalman filter for nonlinear estimation". In: *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pp. 153–158. DOI: 10.1109/ASSPCC.2000.882463.

Weigel, Thilo and Nebel, Bernhard (2003). "RoboCup 2002: Robot Soccer World Cup VI". In: ed. by Gal A. Kaminka, Pedro U. Lima, and Raúl Rojas. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. KiRo – An Autonomous Table Soccer Player, pp. 384–392. DOI: 10.1007/978-3-540-45135-8_34.

Wu, Jun, Wang, Jinsong, and You, Zheng (2010). "An overview of dynamic parameter identification of robots". In: *Robotics and Computer-Integrated Manufacturing* 26.5, pp. 414–419. DOI: http://dx.doi.org/10.1016/j.rcim.2010.03.013.

Xinjilefu, X., Feng, S., and Atkeson, C. G. (May 2016). "A distributed MEMS gyro network for joint velocity estimation". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1879–1884. DOI: 10.1109/ICRA.2016.7487334.

Yamaguchi, N. and Mizoguchi, H. (July 2003). "Robot vision to recognize both face and object for human-robot ball playing". In: *Advanced Intelligent Mechatronics, 2003. AIM 2003. Proceedings. 2003 IEEE/ASME International Conference on*. Vol. 2, 999–1004 vol.2. DOI: 10.1109/AIM.2003.1225478.

Yamakawa, Takeshi et al. (1989). "Applications of Fuzzy Logic Control to Industry Fuzzy controlled robot arm playing two-dimensional ping-pong game". In: *Fuzzy Sets and Systems* 32.2, pp. 149–159. DOI: http://dx.doi.org/10.1016/0165-0114(89)90251-0.

Yang, P. et al. (Dec. 2010). "Control system design for a 5-DOF table tennis robot". In: *Control Automation Robotics Vision (ICARCV), 2010 11th International Conference on*, pp. 1731–1735. DOI: 10.1109/ICARCV.2010.5707252.

Zeilinger, M.N., Morari, M., and Jones, C.N. (May 2014). "Soft Constrained Model Predictive Control With Robust Stability Guarantees". In: *Automatic Control, IEEE Transactions on* 59.5, pp. 1190–1202. DOI: 10.1109/TAC.2014.2304371.

ZMachines (Jan. 2015). *The Official Authentic Page of Z Machines : A Robotic Rock Band*. URL: https://www.facebook.com/ZMachines/info/?entry_point=page_nav_about_item&tab=page_info (visited on 08/23/2016).

A

Rotations and Transformations

A.1. Transformation Matrices

A transformation is given by

$$ {}^{to}_{from}\boldsymbol{T} = \begin{pmatrix} {}^{to}_{from}\boldsymbol{R} & {}^{to}_{from}\boldsymbol{t} \\ \boldsymbol{0} & 1 \end{pmatrix} \quad . \tag{A.1} $$

The translation is defined by the vector ${}^{to}_{from}\boldsymbol{t}$ and the rotation is ${}^{to}_{from}\boldsymbol{R}$. A position vector ${}^{i}\boldsymbol{v}$ is made affine by adding a fourth row with value one such that ${}^{i}\tilde{\boldsymbol{v}} = \begin{pmatrix} v_x & v_y & v_z & 1 \end{pmatrix}^T$ to transform this vector to a new system. For more details the book of Craig (2005) is recommended on how to define transformations.

We start at the EOF coordinate frame and move iteratively to the world coordinate system (WCS):

$$ {}^{\mathrm{w}}_{\mathrm{EOF}}\boldsymbol{T} = {}^{\mathrm{J1}}_{\mathrm{w}}\boldsymbol{T}^{-1}\, {}^{\mathrm{J1'}}_{\mathrm{J1}}\boldsymbol{T}^{-1}\, {}^{\mathrm{J2}}_{\mathrm{J1}}\boldsymbol{T}^{-1}\, {}^{\mathrm{J2'}}_{\mathrm{J2}}\boldsymbol{T}^{-1}\, {}^{\mathrm{J3}}_{\mathrm{J2}}\boldsymbol{T}^{-1}\, {}^{\mathrm{J3'}}_{\mathrm{J3}}\boldsymbol{T}^{-1}\, {}^{\mathrm{EOF}}_{\mathrm{J3}}\boldsymbol{T}^{-1} \tag{A.2} $$

$$ = {}^{\mathrm{w}}_{\mathrm{J1}}\boldsymbol{T}\, {}^{\mathrm{J1}}_{\mathrm{J1'}}\boldsymbol{T}\, {}^{\mathrm{J1}}_{\mathrm{J2}}\boldsymbol{T}\, {}^{\mathrm{J2}}_{\mathrm{J2'}}\boldsymbol{T}\, {}^{\mathrm{J2}}_{\mathrm{J3}}\boldsymbol{T}\, {}^{\mathrm{J3}}_{\mathrm{J3'}}\boldsymbol{T}\, {}^{\mathrm{J3}}_{\mathrm{EOF}}\boldsymbol{T} \quad . \tag{A.3} $$

The $^{-1}$ denotes the inverse matrix. With

$$ {}^{\mathrm{w}}_{\mathrm{J1}}\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0\,\mathrm{mm} \\ 0 & 1 & 0 & 0\,\mathrm{mm} \\ 0 & 0 & 1 & 779.8\,\mathrm{mm} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad , \tag{A.4} $$

$$ {}^{\mathrm{J1}}_{\mathrm{J2}}\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0\,\mathrm{mm} \\ 0 & 0 & 1 & -171.4\,\mathrm{mm} \\ 0 & -1 & 0 & 233.4\,\mathrm{mm} \\ 0 & 0 & 0 & 1 \end{pmatrix} , \tag{A.5} $$

$$ {}^{\mathrm{J2}}_{\mathrm{J3}}\boldsymbol{T} = \begin{pmatrix} 0 & 0 & -1 & 54.9\,\mathrm{mm} \\ 0 & 1 & 0 & 0\,\mathrm{mm} \\ 1 & 0 & 0 & 171.4\,\mathrm{mm} \\ 0 & 0 & 0 & 1 \end{pmatrix} , \tag{A.6} $$

$$ {}^{\mathrm{J3}}_{\mathrm{EOF}}\boldsymbol{T} = \begin{pmatrix} 0 & 1 & 0 & 0\,\mathrm{mm} \\ 0 & 0 & -1 & -1010.069\,\mathrm{mm} \\ -1 & 0 & 0 & 54.9\,\mathrm{mm} \\ 0 & 0 & 0 & 1 \end{pmatrix} . \tag{A.7} $$

The ${}^{i'}_{i}\boldsymbol{T}$ are the joints rotations as defined over their z-axis,

$$ {}^{i'}_{i}\boldsymbol{T} = \begin{pmatrix} \cos(q_i) & \sin(q_i) & 0 & 0 \\ -\sin(q_i) & \cos(q_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} . \tag{A.8} $$

Where i denotes the coordinate frame of joint i before and after i' the rotation of angle q_i. The inverse of this rotation equals its transpose, because of the orthogonality.

Plugging the transformations into Equation A.3 results to the kinematics

$$ \tilde{f}_{\mathrm{kin}}(\boldsymbol{q}) = {}^{\mathrm{w}}_{\mathrm{EOF}}\boldsymbol{T} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} (\mathrm{c}_1\mathrm{s}_2\mathrm{c}_3 - \mathrm{s}_1\mathrm{s}_3)1010.069\,\mathrm{mm} \\ (\mathrm{s}_1\mathrm{s}_2\mathrm{c}_3 + \mathrm{c}_1\mathrm{s}_3)1010.069\,\mathrm{mm} \\ 1010.069\,\mathrm{mm}\,\mathrm{c}_2\mathrm{c}_3 + 1013.2\,\mathrm{mm} \\ 1 \end{pmatrix} . \tag{A.9} $$

A.2. Modified Transformations

Inserting two joints for the bearing leads to a new transformation from WCS to end-effector (EOF), where only the transformation ${}^{\mathrm{w}}_{\mathrm{J1}}\boldsymbol{R}$ changes.

$$ {}^{\mathrm{w}}_{\mathrm{JB1}}\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0\,\mathrm{mm} \\ 0 & 0 & 1 & 0\,\mathrm{mm} \\ 0 & -1 & 0 & 779.8\,\mathrm{mm} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.10} $$

$$ {}^{\mathrm{JB1}}_{\mathrm{JB2}}\boldsymbol{T} = \begin{pmatrix} 0 & 0 & -1 & 0\,\mathrm{mm} \\ 0 & 1 & 0 & 0\,\mathrm{mm} \\ 1 & 0 & 0 & 0\,\mathrm{mm} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.11} $$

$$ {}^{\mathrm{JB2}}_{\mathrm{J1}}\boldsymbol{T} = \begin{pmatrix} 0 & 1 & 0 & 0\,\mathrm{mm} \\ 0 & 0 & -1 & 0\,\mathrm{mm} \\ -1 & 0 & 0 & 0\,\mathrm{mm} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.12} $$

(A.13)

A.3. Rotations for the Extended Calibration Model

$$ {}^{\mathrm{J1}}_{\dot{\boldsymbol{q}}}\boldsymbol{R} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \tag{A.14} $$

$$ {}^{\mathrm{J1}}_{\dot{\boldsymbol{q}}_{\mathrm{fb}}}\boldsymbol{R} = \begin{pmatrix} \mathrm{s}_1\mathrm{c}_{\mathrm{b}2} & -\mathrm{c}_1 \\ \mathrm{c}_1\mathrm{c}_{\mathrm{b}2} & \mathrm{s}_1 \\ \mathrm{s}_{\mathrm{b}2} & 0 \end{pmatrix} \tag{A.15} $$

$$ {}^{\mathrm{EOF}}_{\dot{\boldsymbol{q}}}\boldsymbol{R} = \begin{pmatrix} -\mathrm{s}_2 & 0 & -1 \\ -\mathrm{c}_2\mathrm{s}_3 & \mathrm{c}_3 & 0 \\ \mathrm{c}_2\mathrm{c}_3 & \mathrm{s}_3 & 0 \end{pmatrix} \tag{A.16} $$

$$ {}^{\mathrm{EOF}}_{\dot{\boldsymbol{q}}_{\mathrm{fb}}}\boldsymbol{R} = \begin{pmatrix} \mathrm{c}_{\mathrm{b}2}\mathrm{s}_1\mathrm{c}_2 - \mathrm{s}_{\mathrm{b}2}\mathrm{s}_2 & -\mathrm{c}_1\mathrm{c}_2 \\ \mathrm{c}_{\mathrm{b}2}\mathrm{c}_1\mathrm{c}_3 - \mathrm{c}_{\mathrm{b}2}\mathrm{s}_1\mathrm{s}_2\mathrm{s}_3 - \mathrm{s}_{\mathrm{b}2}\mathrm{c}_2\mathrm{s}_3 & \mathrm{s}_1\mathrm{c}_3 + \mathrm{c}_1\mathrm{s}_2\mathrm{s}_3 \\ \mathrm{c}_{\mathrm{b}2}\mathrm{c}_1\mathrm{s}_3 + \mathrm{c}_{\mathrm{b}2}\mathrm{s}_1\mathrm{s}_2\mathrm{c}_3 - \mathrm{s}_{\mathrm{b}2}\mathrm{c}_2\mathrm{c}_3 & \mathrm{s}_1\mathrm{s}_3 - \mathrm{c}_1\mathrm{s}_2\mathrm{c}_3 \end{pmatrix} \tag{A.17} $$