CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Electrical Engineering
Department of Control Engineering

Modeling and Control of Two-Legged Wheeled Robot

Master's thesis

Adam Kollarčík

Supervisor
Ing. Martin Gurtner

2021

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Kollarčík Adam**	Personal ID number:	**457181**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		
Branch of study:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Modeling and control of two-legged wheeled robot

Master’s thesis title in Czech:

Modelování a řízení dvounohého kolového robotu

Guidelines:

The goal of this thesis is to design a control system for a two-legged wheeled robot. The control system is required to balance the robot, steer the robot, and perform a jump forward. For that purpose, a simplified mathematical model of the robot is expected to be derived. The control system's capability should be demonstrated in a physics-based simulator with a high-fidelity 3D dynamical robot model.
1) Create a mathematical model of a planar version of the robot.
2) Design a controller stabilizing the robot in the erected position, both with the fixed and varying leg lengths. Extend the controller by the capability of positioning the robot.
3) Using the planar mathematical model, find a trajectory that gets the robot off the floor. Test this trajectory on the full 3D model in a simulation environment.
5) Demonstrate the stabilizing controller on the real robot.

Bibliography / sources:

[1] Klemm, Victor, et al. LQR-Assisted Whole-Body Control of a Wheeled Bipedal Robot with Kinematic Loops. IEEE Robotics and Automation Letters, 2020, 5.2: 3745-3752.
[2] Posa, Michael; CANTU, Cecilia; TEDRAKE, Russ. A direct method for trajectory optimization of rigid bodies through contact. The International Journal of Robotics Research, 2014, 33.1: 69-81.
[3] Featherstone, Roy. Rigid body dynamics algorithms. Springer, 2014.

Name and workplace of master’s thesis supervisor:

Ing. Martin Gurtner, Department of Control Engineering, FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **18.09.2020** Deadline for master's thesis submission: **05.01.2021**

Assignment valid until:
by the end of winter semester 2021/2022

Ing. Martin Gurtner	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Adam Kollarčík
January 2021

Acknowledgments

I would like to thank my supervisor *Martin Gurtner*, for his guidance, help, and patience. Without him, this thesis would not be possible, and I am incredibly grateful that I could cooperate on this task with him. Furthermore, I am deeply indebted to *Zdeněk Hurák*, who invited me to the AA4CC research group and gave me the opportunity to work on exciting projects.

I could not forget my friends and colleagues. Namely, I would like to thank *Krištof Pučejdl*, for his help and for designing and constructing the robot. You have done a great job! Special thanks to *Loi Do*, for providing me this template, for his advice, and for being the best climbing partner. I very much appreciate my friend *Vašek Pritzl*, for proofreading, and for sending me rubber chicken song cover videos[1]. I also had the pleasure of sharing the lab and a lot of funny moments with *Josef Matouš*, *Lukáš Černý*, and *Adam Polák*. Thank you!

Finally, I would like to thank my family for their limitless and unprecedented love and support. Without them, I would not have made it this far.

[1] `https://www.youtube.com/c/RomeroMay`

Abstract

This thesis focuses on modeling and control of a bipedal wheeled robot. The main objective is threefold: provide simulation tools essential for control design, design a control system capable of balancing and steering the robot, and find a feasible jump trajectory. First, a non-linear planar model of the robot is derived using Euler-Lagrange equations. In addition, a 3D simulation environment is created within the Gazebo robotics simulator. The 3D simulations are used throughout the thesis for verification and testing purposes. Then, the controller consisting of two independent parts is designed. The wheels are controlled by a linear-quadratic regulator (LQR) with integral control based on a linearized two-wheel inverted pendulum 3D model. Two proportional-derivative (PD) regulators are used to control the legs of the robot. Furthermore, Unscented Kalman Filter (UKF) for state estimation and mainly the optimization-based method for jump trajectory computation are implemented. Lastly, the LQR and PD controllers were tested on the real robot, and the results from the conducted experiments are also documented in this thesis.

Keywords: bipedal wheeled robot, Gazebo, LQR, jump trajectory computation, UKF

Abstrakt

Tato práce se zabývá modelováním a řízením dvounohého robotu s koly. Cílem práce je vytvořit potřebné simulační nástroje pro návrh řídicího systému, návrh řídicího systému schopného stabilizace a řízení pohybu robotu, a nalézt vhodné trajektorie skoku. Nejdříve je odvozen nelineární planární model robotu za použití Euler-Lagrangeových rovnic. Navíc je k tomu vytvořeno 3D simulační prostředí v robotickém simulátoru Gazebo. Tyto 3D simulace jsou napříč prací použity jako hlavní verifikační a testovací nástroj. Poté je navržen řídicí systém skládající se ze dvou částí. Kola jsou řízena lineárně-kvadratickým regulátorem (LQR) s integrálním řízením navrženým na základě linearizovaného 3D modelu dvojkolého inverzního kyvadla. Nohy jsou řízeny dvěma proporčně-derivačními (PD) regulátory. Dále je implementován tzv. Unscented Kalman Filter (UKF) k odhadování stavů, a také na optimalizaci založená metoda určena k výpočtu trajektorie skoku. Navržené LQR a PD regulátory byly otestovány na reálném robotu a výsledky experimentů jsou zde také prezentovány.

Klíčová slova: dvounohý robot s koly, Gazebo, LQR, výpočet trajektorie skoku, UKF

Contents

1 | Introduction

In this thesis, I deal with modeling, simulation, and control of a two-legged wheeled robot. I present a simplified planar model of the robot, a control system, and a jump trajectory computation method. I verified all the components in a physics-based simulator and successfully used the designed controller for real robot stabilization and steering. The results are documented by a video[1] and the code is published in a gitlab repository[2].

1.1 Problem Statement

The objective of this thesis is threefold. Firstly, to create an environment for the 3D simulation of the robot within a robotic simulation software. Secondly, to design a simple control system that is capable of balancing and steering the robot. Finally, to derive a planar model of the robot and use it for jump trajectory computation. These components are intended to form basic simulation and control tools for the robot that my colleagues designed and whose prototype was being manufactured simultaneously to work on this thesis. I was also able to test the designed controller on the real robot, thanks to my fellow workers that managed to finish the prototype at the beginning of December, despite the second wave of the COVID-19 outbreak.

1.2 Related Work

With the increasing popularity of mobile robotics in recent years, the number of various original legged systems also rises. Undoubtedly, the most famous are the robots made by Boston Dynamics[3], namely the human-like Atlas, four-legged Spot, and two-legged wheeled Handle. Among others, I want to mention the *ANYmal* [1], [2], and mainly the *Ascento* [3], [4] which served as inspiration for our robot sharing the same topology.

For the ANYmal and in the newest Ascento paper [4] they implemented so-called *Whole-Body Control* (WBC) [5] which is a modern real-time optimization-based control approach requiring a full model of the robot. Even though the WBC is a well-performing method used in many legged robot projects, it does not fit the goal of this thesis because of its design and implementation complexity. In the first Ascento paper [3], authors used LQR based on

[1]https://youtu.be/B6OaJCYD5C8

[2]https://gitlab.fel.cvut.cz/kollaada/master-thesis-by-adam-kollarcik

[3]https://www.bostondynamics.com/

linearized inverted pendulum model and achieved good results, especially if the simplicity is taken into consideration. This is the approach I chose as well.

1.3 Robot Description

At the time of writing the thesis, the robot is not documented anywhere. Thus, I here briefly describe the construction and electronics aspects that are relevant to this thesis. I want to emphasize that *I was not* involved in any design, construction, or programming tasks, and everything described in this section is the work of my colleagues.

As I already mentioned, our robot captured in Figure 1.1 shares the same topology with Ascento, having two wheeled legs with a closed kinematic chain. In each knee, there is a torsion spring placed to counteract the weight in the balancing position. Also, most of the parts were created using a 3D printer, including the tires. The dimensions and other parameters listed in Chapter 2 differ from the actual values because they were changing during the design process or were not identified.

The simplified diagram of the used electronics hierarchy is shown in Figure 1.2. The robot has four *eX8108 105KV* brushless DC motors. Two are placed in wheels, and the remaining two are placed in the joints connecting the black links with the body on both sides, together

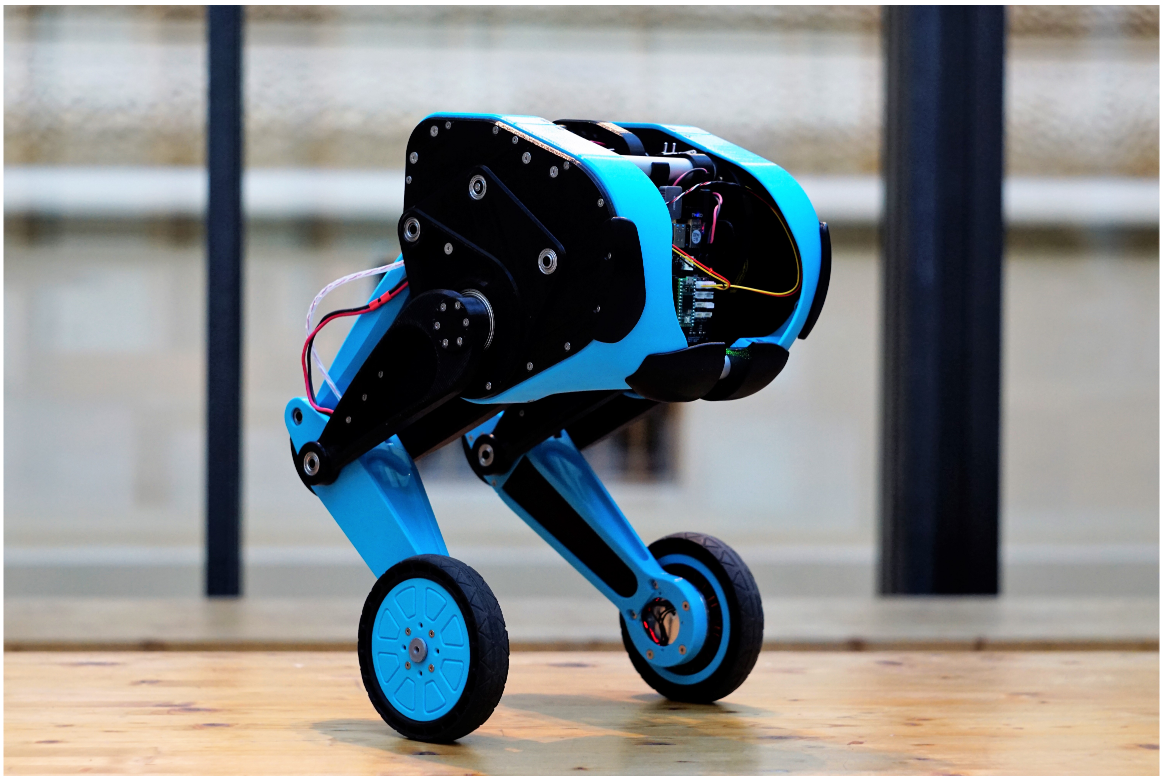


Figure 1.1: First working prototype of our bipedal wheeled robot.

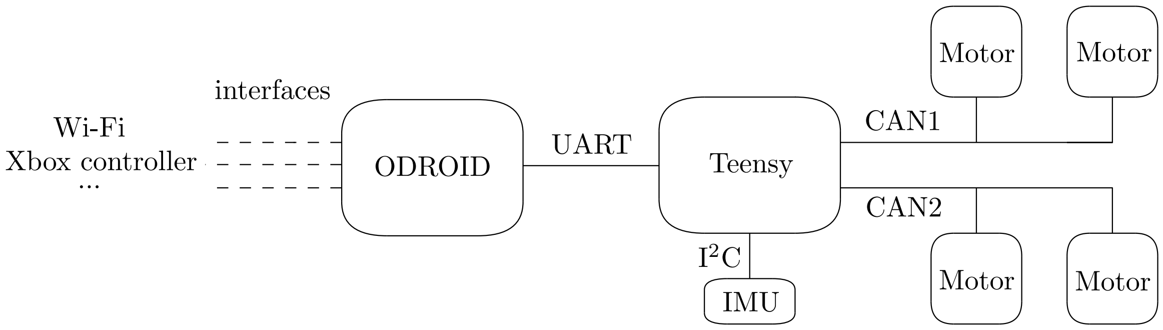


Figure 1.2: Electronics diagram.

with 1:16.5 gearboxes that increase the maximal achievable torques. The motors are driven by four 3-phase controllers[4] at 40 kHz, providing position, velocity, and current feedback together with corresponding measurements. The robot control loop runs at 1 kHz on *Teensy 4.0*, connected with the drivers through CAN. The Teensy sends position, velocity, and current (torque) references and PD controller parameters to the drivers in each iteration. Additionally, the *ICM-42688-P* IMU measures linear accelerations and angular velocities of the body at 2 kHz. These measurements are filtered with a low-pass filter with 1 kHz cut-off frequency and sent to the Teensy through I^{2}C. Finally, *ODROID N2+* communicating with Teensy through UART at 2 Mbaud/s serves for other computations and interfacing. The whole system is powered by a commonly available 24V cordless power tool battery.

Based on the hardware, the following specifications were formulated:

1. The sampling frequency of the control loop is 1 kHz.
2. The measured quantities are the motor positions, motor velocities, motor torques, body pitch angle, body roll angle, body pitch rate, body roll rate, and body yaw rate.
3. The motors should not exert torque higher than 2 Nm.

1.4 Outline

This thesis is divided into five chapters. In Chapter 2, I derive the simplified planar model of the robot and comment on the creation of the 3D simulation environment in the Gazebo simulator. Chapter 3 is devoted to the control system. I describe the inverted pendulum-based LQR design, the Unscented Kalman Filter design, and the optimization-based jump trajectory computation. In Chapter 4, I present the real robot experiments. I conclude the thesis in Chapter 5 by discussing the achieved results and by suggesting future improvements.

[4]The motor controllers are based on Ben Katz's design: `https://github.com/bgkatz/3phase_integrated`

2 | Modeling and Simulation

In this chapter, I derive a simplified planar model. Also, I present a full 3D robot simulation model within a robotics simulation software that served for testing and verification purposes.

2.1 Simplified Planar Model

The planar variant of the robot is shown in Figure 2.1a. It consists of a wheel, a body, and three links denoted by their lengths l_1, l_2, l_3. The parts are connected with five revolute joints. Two motors placed in the wheel and in the connection between the body and the second link generate torques u_0 and u_4. Also, a torsion spring is attached between the first and the second link. All the parameters are listed in Table 2.1.

I approached the modeling in two steps similarly as in [4]. First, I derived a robot model without the fifth joint attached, as shown in Figure 2.1b using the Euler-Lagrange equations. Second, I added the reaction forces (torques) caused by fixing the third link to the body with the Lagrange multiplier framework. Lastly, I converted the obtained differential-algebraic equations (DAE) model to a reduced-order ordinary differential equation (ODE) model because it is more convenient to work with as most of the standard control system design techniques require ODE formulation.

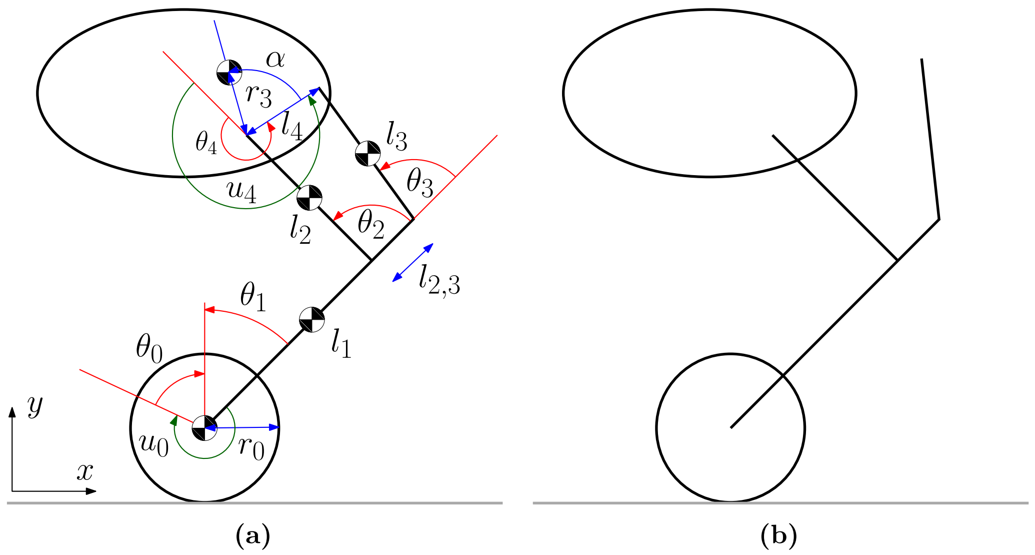


Figure 2.1: Planar wheeled robot: **(a)** Diagram with labeled quantities. **(b)** Basic diagram with unconnected fifth joint.

Table 2.1: Planar robot model parameters

Symbol	Parameter	Value	Unit
b	damping coefficient	0.01	$\mathrm{N\,m\,s\,rad^{-1}}$
I_0	moment of inertia of the wheel	$7.35 \cdot 10^{-4}$	$\mathrm{kg\,m^2}$
I_1	moment of inertia of the first link	$9.60 \cdot 10^{-4}$	$\mathrm{kg\,m^2}$
I_2	moment of inertia of the second link	$5.89 \cdot 10^{-4}$	$\mathrm{kg\,m^2}$
I_3	moment of inertia of the third link	$3.10 \cdot 10^{-4}$	$\mathrm{kg\,m^2}$
I_4	moment of inertia of the body	$1.19 \cdot 10^{-2}$	$\mathrm{kg\,m^2}$
k	torsion spring stiffness	0.25	$\mathrm{N\,m\,rad^{-1}}$
l_1	length of the first link	0.24	m
l_2	length of the second link	$1.88 \cdot 10^{-1}$	m
l_3	length of the third link	$1.93 \cdot 10^{-1}$	m
$l_{2,3}$	second and third joint distance	0.05	m
l_4	fourth and fifth joint distance	$0.93 \cdot 10^{-1}$	m
m_0	mass of the wheel	0.30	kg
m_1	mass of the first link	0.20	kg
m_2	mass of the second link	0.20	kg
m_3	mass of the third link	0.10	kg
m_4	mass of the body	1.50	kg
r_0	wheel radius	0.07	m
r_3	body COG and fourth joint distance	0.05	m
α	rotation offset of the body	45	$^\circ$
β	torsion spring zero torque angle	-487	$^\circ$

2.1.1 Potential and Kinetic Energy

The open-chain robot has five degrees of freedom, assuming the wheel does not slip and is always in contact with the ground. The generalized coordinates completely describing the position of the system are the wheel and joint angles

$$\boldsymbol{\theta} = \begin{bmatrix}\theta_0 & \theta_1 & \theta_2 & \theta_3 & \theta_4\end{bmatrix}^\intercal . \tag{2.1}$$

Using coordinate transformations, I obtained the center of gravity (COG) positions of the wheel, body, and links (assuming that the links have their COGs located in the geometrical center) in an inertial frame of reference:

$$\boldsymbol{x}_0 = \boldsymbol{x}_0(\theta_0)\,, \tag{2.2}$$

$$\boldsymbol{x}_1 = \boldsymbol{x}_1(\theta_0, \theta_1)\,, \tag{2.3}$$

$$\boldsymbol{x}_2 = \boldsymbol{x}_2(\theta_0, \theta_1, \theta_2)\,, \tag{2.4}$$

$$\boldsymbol{x}_3 = \boldsymbol{x}_3(\theta_0, \theta_1, \theta_3)\,, \tag{2.5}$$

$$\boldsymbol{x}_4 = \boldsymbol{x}_4(\theta_0, \theta_1, \theta_2, \theta_4)\,. \tag{2.6}$$

The potential energy V is

$$V = \sum_{i=0}^{4} m_i y_i g + \frac{1}{2} k (\theta_2 - \beta)^2 \,, \tag{2.7}$$

where m_i is the mass of the ith component, y_i is the y-coordinate of the COG of the i-th component, and g is the gravitational acceleration. Stiffness k and zero force angle β are parameters of the torsion spring. The kinetic energy T is

$$T = \sum_{i=0}^{4} \left(\frac{1}{2} m_i \dot{\boldsymbol{x}}_i^\intercal \dot{\boldsymbol{x}}_i + \frac{1}{2} I_i \dot{\omega}_i^2 \right) \,, \tag{2.8}$$

where I_i is a moment of inertia for rotation around COG of the ith component, $\dot{\boldsymbol{x}}$ is the time derivative of the COG position of the ith component, and ω_i is the angular velocity of the corresponding link

$$\begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta}_0 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \end{bmatrix} . \tag{2.9}$$

According to [6], I reformulated the kinetic energy as follows:

$$T = \frac{1}{2} \dot{\boldsymbol{\theta}}^\intercal \boldsymbol{M} \dot{\boldsymbol{\theta}} \,, \quad \dot{\boldsymbol{\theta}} = \begin{bmatrix} \dot{\theta}_0 & \dot{\theta}_1 & \dot{\theta}_2 & \dot{\theta}_3 & \dot{\theta}_4 \end{bmatrix}^\intercal \,, \tag{2.10}$$

where $\boldsymbol{M} = \boldsymbol{M}(\boldsymbol{\theta})$ is the 5×5 *mass matrix*, which depends only on the angles.

2.1.2 Equations of Motion

After formulating the potential and kinetic energy, I proceeded in deriving the motion equations. They can be expressed in a vector form as:

$$\boldsymbol{\tau}_{\text{in}} = \boldsymbol{M} \ddot{\boldsymbol{\theta}} + (\boldsymbol{C} + \mathbf{D}) \dot{\boldsymbol{\theta}} + \boldsymbol{g} \,, \tag{2.11}$$

where $\boldsymbol{M}$ is the mass matrix mentioned above, $\mathbf{D}$ is the matrix of friction and damping coefficients computed later, $\boldsymbol{C} = \boldsymbol{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})$ is the 5×5 *Coriolis matrix*, $\boldsymbol{\tau}_{\text{in}} = \begin{bmatrix} u_0 & u_0 & 0 & 0 & u_4 \end{bmatrix}^\intercal$ is the input torque vector, and

$$\boldsymbol{g}(\boldsymbol{\theta}) = \frac{\partial V}{\partial \boldsymbol{\theta}} \,. \tag{2.12}$$

I obtained the Coriolis matrix using the *Christoffel symbols of the first kind* Γ_{ijk}:

$$\Gamma_{ijk} = \frac{1}{2} \left(\frac{\partial m_{ij}}{\partial \theta_k} + \frac{\partial m_{ik}}{\partial \theta_j} - \frac{\partial m_{jk}}{\partial \theta_i} \right) \,, \tag{2.13}$$

$$c_{ij} = \sum_{k=0}^{4} \Gamma_{ijk} \dot{\theta}_k \,, \tag{2.14}$$

where m_{ik}, m_{ik}, m_{jk} are the $(i+1, j+1)$th, the $(i+1, k+1)$th, and the $(j+1, k+1)$th element[1] of the mass matrix, respectively. Similarly, c_{ij} is the $(i+1, j+1)$th element of the Coriolis matrix.

2.1.3 Adding Constraints

The geometrical constraints caused by fixing the fifth joint in place are:

$$\boldsymbol{a} = \begin{bmatrix} l_2 \cos(\theta_2) + l_4 \cos(\theta_2 + \theta_4) - l_{2,3} - l_3 \cos(\theta_3) \\ l_2 \sin(\theta_2) + l_4 \sin(\theta_2 + \theta_4) - l_3 \sin(\theta_3) \end{bmatrix} = \mathbf{0}\,. \tag{2.15}$$

The time derivative of the constraints should also be zero, thus

$$\frac{\mathrm{d}\boldsymbol{a}}{\mathrm{d}t} = \dot{\boldsymbol{a}} = \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{\theta}} \dot{\boldsymbol{\theta}} = \boldsymbol{A}\dot{\boldsymbol{\theta}} = \mathbf{0}\,. \tag{2.16}$$

The same is true for the second time derivative as well:

$$\frac{\mathrm{d}\dot{\boldsymbol{a}}}{\mathrm{d}t} = \dot{\boldsymbol{A}}\dot{\boldsymbol{\theta}} + \boldsymbol{A}\ddot{\boldsymbol{\theta}} = \mathbf{0}\,. \tag{2.17}$$

Under the assumption that $\boldsymbol{a}$ are workless constraints, the following must hold for the reaction forces $\boldsymbol{\tau}_{\text{con}}$ caused by them:

$$\boldsymbol{\tau}_{\text{con}}^{\mathsf{T}} \dot{\boldsymbol{\theta}} = \mathbf{0}\,. \tag{2.18}$$

By comparing (2.16) and (2.18) I can say that the reaction forces are linear combinations of the rows of $\boldsymbol{A}$. I write this as

$$\boldsymbol{\tau}_{\text{con}} = \boldsymbol{A}^{\mathsf{T}} \boldsymbol{\lambda}\,, \tag{2.19}$$

where elements of vector $\boldsymbol{\lambda}$ are usually referred to as *Lagrange multipliers*. I can now expand the equations of motion of the unconstrained system (2.11) with (2.17) and (2.19) to obtain the dynamic equations for the robot with closed kinematic chain:

$$\boldsymbol{\tau}_{\text{in}} + \boldsymbol{A}^{\mathsf{T}} \boldsymbol{\lambda} = \boldsymbol{M}\ddot{\boldsymbol{\theta}} + (\boldsymbol{C} + \mathbf{D})\dot{\boldsymbol{\theta}} + \boldsymbol{g}\,, \tag{2.20}$$

$$\dot{\boldsymbol{A}}\dot{\boldsymbol{\theta}} + \boldsymbol{A}\ddot{\boldsymbol{\theta}} = \mathbf{0}\,. \tag{2.21}$$

2.1.4 Damping

After fixing the fifth joint in place, matrix $\mathbf{D}$ can be computed. The interior angles of a quadrilateral add up to 2π rad, therefore the norm of the angular velocity of the fifth joint is

$$\left|\dot{\theta}_5\right| = \left|\dot{\theta}_2 - \dot{\theta}_3 + \dot{\theta}_4\right|. \tag{2.22}$$

[1] It is needed to add one to the element's position since the first index is 0.

Assuming the same linear damping b in all joints, the dissipation torques $\boldsymbol{d}$ are

$$\boldsymbol{d} = -\mathbf{D}\dot{\boldsymbol{\theta}} = -\frac{\partial R}{\partial \dot{\boldsymbol{\theta}}}\,, \tag{2.23}$$

where R is the dissipated energy also referred to as the *Rayleigh's Dissipation Function* [7]:

$$R(\dot{\boldsymbol{\theta}}) = \frac{1}{2}b\left[(\dot{\theta}_0 + \dot{\theta}_1)^2 + \dot{\theta}_2^2 + \dot{\theta}_3^2 + \dot{\theta}_4^2 + \dot{\theta}_5^2\right]\,. \tag{2.24}$$

I obtained matrix $\boldsymbol{D}$ by combining equations (2.22), (2.23), and (2.24):

$$\mathbf{D} = b\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & -1 & 1 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & -1 & 2 \end{bmatrix}\,. \tag{2.25}$$

2.1.5 Reduced Order Model

The pseudo velocity approach [8] is an effective way of converting the derived DAE model (2.20), (2.21) to an ODE model. If I define $\boldsymbol{G}(\boldsymbol{\theta})$ as a 5×3 matrix, where columns are the basis of the null space of $\boldsymbol{A}$

$$\boldsymbol{A}\boldsymbol{G} = \boldsymbol{0}\,, \tag{2.26}$$

then correspondingly to (2.18) I can write that

$$\dot{\boldsymbol{\theta}} = \boldsymbol{G}\boldsymbol{v}\,, \tag{2.27}$$

such that $\boldsymbol{v}$ is a vector of three real values called *pseudo velocities*. It is worth mentioning, that even though there is an infinite number of valid matrices $\boldsymbol{G}$ giving various transformations, I found one with simple mapping such as:

$$\boldsymbol{v} = \begin{bmatrix} \dot{\theta}_0 & \dot{\theta}_1 & \dot{\theta}_3 \end{bmatrix}^\mathsf{T} \tag{2.28}$$

This is not surprising, as $\dot{\theta}_0$ and $\dot{\theta}_1$ should not be affected by the constraints and $\dot{\theta}_2$, $\dot{\theta}_3$, $\dot{\theta}_4$ depend on each other.

By multiplying (2.20) with $\boldsymbol{G}^\mathsf{T}$ from left and substituting with (2.27) I obtained

$$\boldsymbol{G}^\mathsf{T}\boldsymbol{\tau}_{\text{in}} = \boldsymbol{M}_g\dot{\boldsymbol{v}} + \boldsymbol{C}_g\boldsymbol{v} + \boldsymbol{G}^\mathsf{T}\boldsymbol{g}\,, \tag{2.29}$$

where

$$\boldsymbol{M}_g = \boldsymbol{G}^\mathsf{T}\boldsymbol{M}\boldsymbol{G}\,, \quad \boldsymbol{C}_g = \boldsymbol{G}^\mathsf{T}\boldsymbol{M}\dot{\boldsymbol{G}} + \boldsymbol{G}^\mathsf{T}\left(\boldsymbol{C} + \mathbf{D}\right)\boldsymbol{G}\,. \tag{2.30}$$

Matrix $\boldsymbol{M}_g$ is positive definite [8], which allows me to simplify (2.29) and receive equation

$$\dot{\boldsymbol{v}} = \boldsymbol{M}_g^{-1} \left(-\boldsymbol{C}_g \boldsymbol{v} - \boldsymbol{G}^\intercal \boldsymbol{g} + \boldsymbol{G}^\intercal \boldsymbol{\tau}_{\text{in}} \right) , \tag{2.31}$$

that together with (2.27), forms the reduced-order ODE model.

2.2 Full Robot 3D Simulation

It is a common and necessary practice to verify a control system's performance using complex model simulations before its real-life application to avoid any unnecessary damage. There are various free and paid software tools offering 3D robot simulations, including the Simscape Multibody toolbox for Matlab/Simulink. Even though it would be appealing to stay within the Matlab/Simulink products that I used for most of the work, I chose the Open Robotics' Gazebo open-source simulator. It has become one of the standard simulation tools in the robotics community used in many prestigious competitions such as *DARPA Subterranean Challenge*[2] and *NASA Space Robotics Challenge*[3].

2.2.1 Gazebo & Simulink Co-Simulation

Gazebo performs simulations through so-called worlds. A world is a simulation environment containing all objects and settings of the simulation. The words can be created and set through the program's graphical user interface or via Simulation Description Format (SDF) file, an XML format file created especially for this purpose. A crucial feature of SDF for me was the support of kinematic loops, which a similar file format URDF lacks.

A model might be created directly in the world it is simulated in or more conveniently imported from a separate SDF model file. The creation of a model in SDF itself is straightforward. First, all links are defined. That means setting their position, orientation, physical parameters, collision mesh, and visual mesh. Then, the links are interconnected with joints. Again, there are multiple parameters and settings options. Also, SDF has a selection of standard sensors such as IMU, GPS or camera that may be directly included in the model. Lastly, the model's functionality can be expanded with C++ plugins that use Gazebo API.

I proceeded exactly as mentioned above[4] while creating the model of the robot that is shown in Figure 2.2. I picked the same parameters as in the simplified model in cases of matching counterparts (see Table 2.1). The only exception was the weight of the body. Since the number of actuators is doubled in the 3D model, I chose the weight as twice the amount used in the simplified 2D case.

[2] `www.subtchallenge.com`

[3] `www.spaceroboticschallenge.com`

[4] All files are available at `https://gitlab.fel.cvut.cz/kollaada/master-thesis-by-adam-kollarcik`.

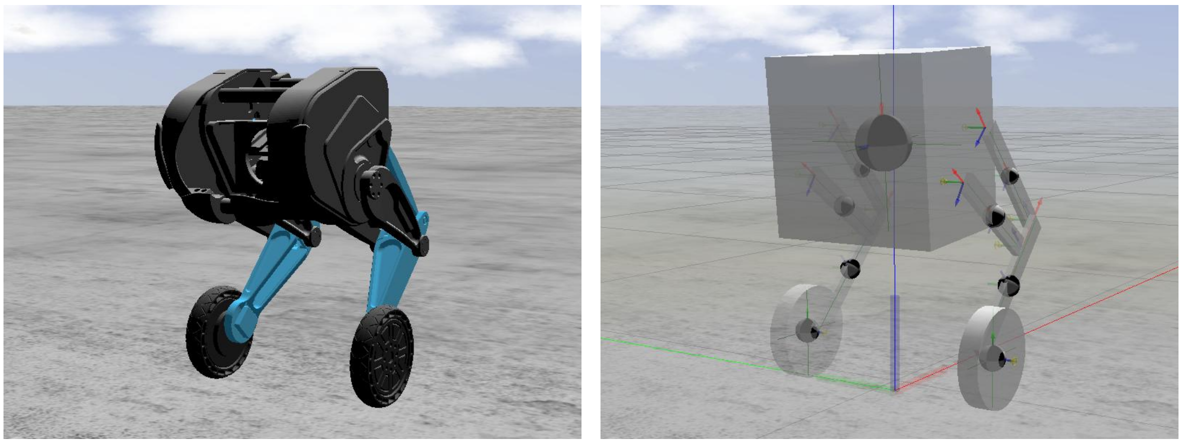


Figure 2.2: Model of the robot in Gazebo simulator: **(a)** Regular view. **(b)** Transparent view of collision meshes, coordinate frames and COGs.

To enable feedback control, I created a model plugin that reads all the measurements and state information. The plugin publishes the data on a specific TCP/IP-based Gazebo communication channel called a topic. The topic is connected to the Simulink environment, as Simulink has Gazebo co-simulation support enabled by the Robotics System Toolbox. The computed control action values are published on a dedicated topic from the Simulink feedback loop. They are then retrieved by the plugin and applied to the model. The simulation diagram is shown in Figure 2.3.

2.2.2 Comparison with the Planar Model

For comparison of 3D simulation with the derived model, I chose two simple open-loop scenarios. In both cases, the useful simulation time for the comparison is small because the planar model does not include ground collisions nor taking off the ground. Unfortunately, this restriction is inevitable in open-loop simulations.

In the first scenario, the same constant torque was applied to the wheel. As can be seen in

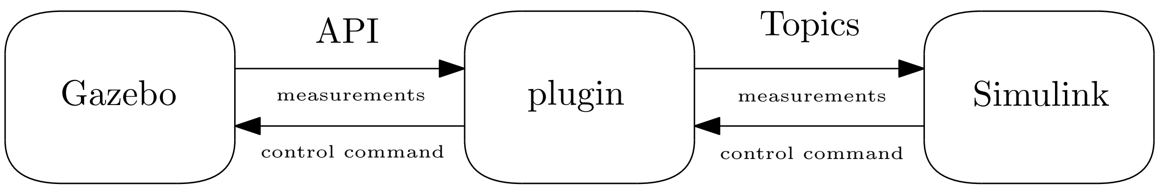


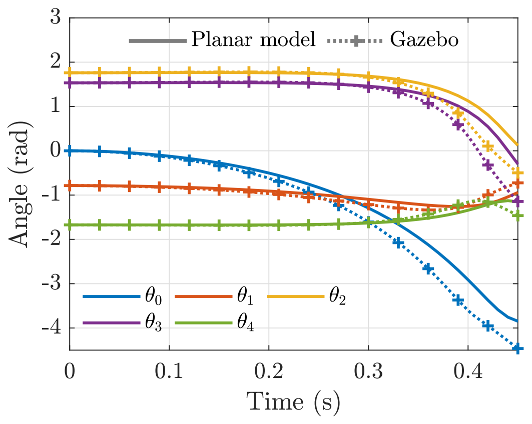
Figure 2.3: Simulation diagram.

Figures 2.4a, 2.4b[5] the robot in the 3D simulation falls down faster. Otherwise, the trend in both cases is similar. At around 0.4 s, the robot fell and lifted the wheels off the ground in Gazebo.

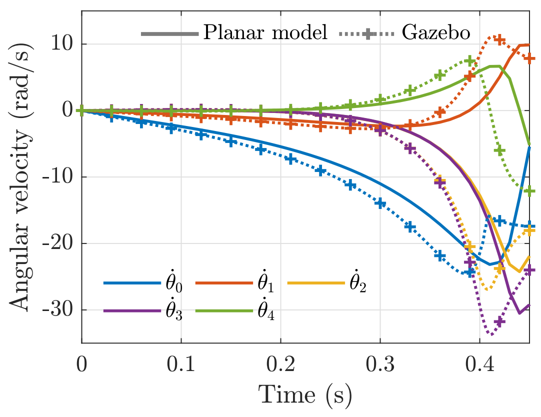
In the second scenario, I applied the same constant torque(s) to the body motor(s). This is shown in Figures 2.4c, 2.4d where both simulations are almost indistinguishable. The robot jumps off the ground in Gazebo slightly before 0.7 s.

According to the results, I found the planar model sufficiently accurate. This is also confirmed in section 3.3, where the precomputed jump sequences match the feedforward behavior of the 3D model.

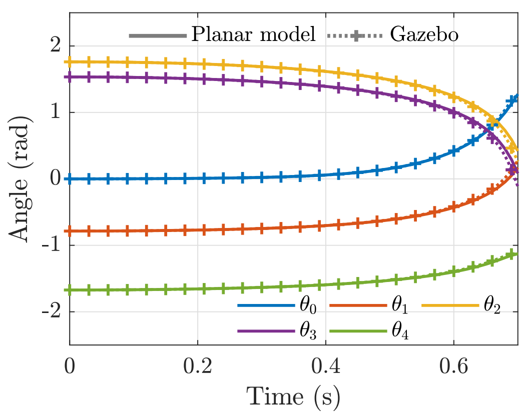
[5]In the figures, only data from one side of the 3D robot are captured, as both sides are the same when equal torques are used.



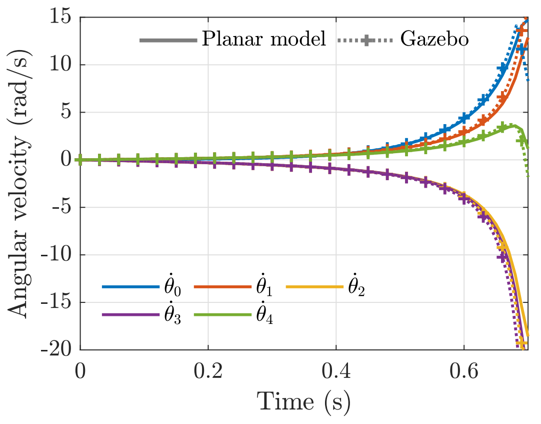
(a)



(b)



(c)



(d)

Figure 2.4: Comparison scenarios: **(a)**,**(b)** First scenario — constant wheel torque(s). **(c)**,**(d)** Second scenario — constant body torque(s).

3 | Control System

In this chapter, I present the controller for stabilization and motion control, the state observer, and computation of jumping trajectory. I designed a state feedback control law based on a linearized 3D wheeled inverted pendulum model. In order to filter and fuse all the available measurements, I used an Unscented Kalman Filter. Finally, I describe the computation of the jumping state trajectory by solving a non-linear optimization problem.

3.1 Stabilization and Motion Control

Initially, I intended to design two independent LQ state controllers for the robot's left and right side based on the linearized planar model. Unfortunately, due to the non-minimum phase dynamics of such a system, this approach turned out to be unstable in the yaw motion as the coupling of the two sides was not considered. I managed to overcome this problem in simulations by introducing an external proportional yaw damper. This solution proved to be non-functional when applied on the real robot since the damper gains that worked in the simulation were too big, which led to twitching and falling of the real robot. On the other hand, the damper with smaller gain values could not stabilize the yaw dynamics. Thus, I had to use a model with the yaw coupling included for the controller design. The natural choice would be a full 3D model of the robot. Nevertheless, this thesis aims to design a control system for the first prototype of a robot. Thus, I intended to keep the mathematical model and the control system as simple as possible. I decided to use an inverse pendulum simplification of the robot for the wheel stabilization and PD control of the body motors, similarly as in [3].

3.1.1 Equilibrium of the Non-Linear Model

Before the controller design itself, I needed to find the robot configuration—the equilibrium—in which I would stabilize the robot. Since the derived motion equations are non-linear and complex, it would be a tedious and maybe even impossible task to compute its solution analytically. Hence, I applied numerical optimization.

I set the angular accelerations and velocities in the equilibrium to zero, therefore according to (2.27), the :

$$\dot{\boldsymbol{v}} = \mathbf{0}\,, \quad \boldsymbol{v} = \mathbf{0}\,. \tag{3.1}$$

This together with the reduced-order model from (2.31) gives the equilibrium condition

$$\boldsymbol{M}_g^{-1}\left(-\boldsymbol{G}^\mathsf{T}\boldsymbol{g}+\boldsymbol{G}^\mathsf{T}\boldsymbol{\tau}_\text{in}\right)=\boldsymbol{0}, \tag{3.2}$$

which can further be simplified to

$$\boldsymbol{G}^\mathsf{T}\left(-\boldsymbol{g}+\boldsymbol{\tau}_\text{in}\right)=\boldsymbol{0}. \tag{3.3}$$

If the term $\boldsymbol{g}$ is compensated by the inputs $\boldsymbol{\tau}_\text{in}$ or if vector $-\boldsymbol{g}+\boldsymbol{\tau}_\text{in}$ is an element of the null space of $\boldsymbol{G}^\mathsf{T}$, the condition is satisfied. To find such equilibrium, I formulated the following optimization task:

$$\underset{\boldsymbol{\theta},\boldsymbol{\tau}_\text{in}}{\arg\min}\,\boldsymbol{\tau}_\text{in}^\mathsf{T}\boldsymbol{\tau}_\text{in}\,, \tag{3.4}$$

minimizing the input torques under conditions (3.3), and closed chain kinematic constraints (2.15) with the specified desired value for θ_1:

$$\begin{bmatrix}\boldsymbol{G}^\mathsf{T}(\boldsymbol{\tau}_\text{in}-\boldsymbol{g})\\ \boldsymbol{a}\\ \theta_1+\frac{\pi}{4}\end{bmatrix}=\boldsymbol{0}\,. \tag{3.5}$$

There are multiple solutions to this problem; some of them lead to link intersection. To avoid those configurations, I restricted the values of the remaining angles such that they lie within the acceptable range. I found the desired solution with a solver for constrained non-linear problems in Matlab, and the results are shown in Table 3.1.

3.1.2 3D Wheeled Inverted Pendulum Model

The models of most pendulum-like systems are well-known, and the wheeled pendulum shown in Figure 3.1 is no exception. Therefore, I did not have to derive it myself and used the one derived in [9]. For the reader's convenience, I repeat the equations here. The state-space model is the following:

$$\begin{bmatrix}\ddot{x}\\ \ddot{\varphi}\\ \ddot{\psi}\end{bmatrix}=\begin{bmatrix}m_{11} & m_{12} & 0\\ m_{21} & m_{22} & 0\\ 0 & 0 & m_{33}\end{bmatrix}^{-1}\left(\begin{bmatrix}\frac{1}{r_0}(u_{0\text{R}}+u_{0\text{L}})\\ m_\text{p}lg\sin\varphi-u_{0\text{R}}-u_{0\text{L}}\\ \frac{w}{2}(u_{0\text{R}}-u_{0\text{L}})\end{bmatrix}-\begin{bmatrix}\frac{2b}{r_0^2} & d_{12} & d_{13}\\ \frac{-2b}{r_0} & 2b & d_{23}\\ d_{31} & d_{32} & d_{33}\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{\varphi}\\ \dot{\psi}\end{bmatrix}\right), \tag{3.6}$$

Table 3.1: Equilibrium angle and input values

$\theta_{0,\text{eq}}$ (rad)	$\theta_{1,\text{eq}}$ (rad)	$\theta_{2,\text{eq}}$ (rad)	$\theta_{3,\text{eq}}$ (rad)	$\theta_{4,\text{eq}}$ (rad)	$u_{0,\text{eq}}$ (Nm)	$u_{4,\text{eq}}$ (Nm)
0.00	−0.79	1.76	1.53	−1.67	0.00	0.00

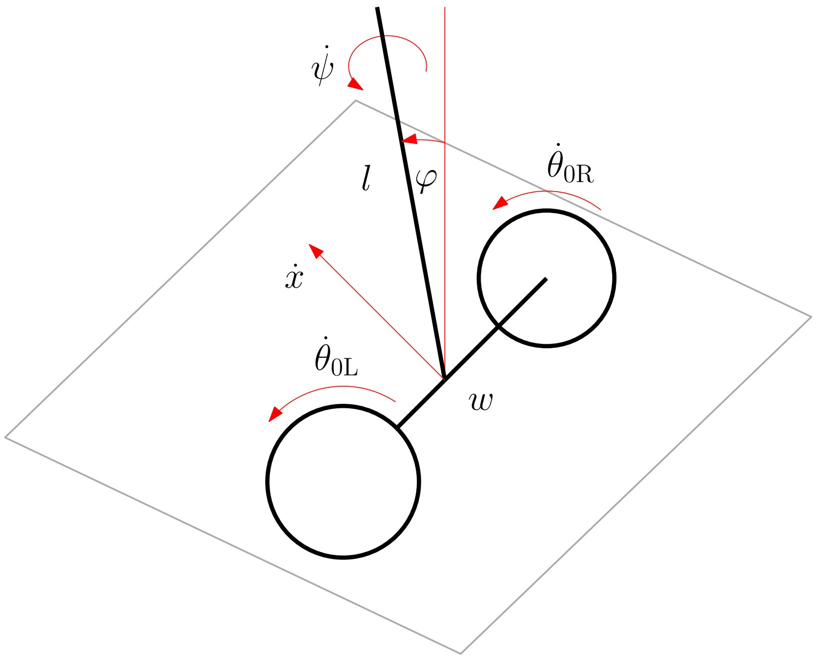


Figure 3.1: Simplified 3D wheeled inverted pendulum

$$m_{11} = m_\mathrm{p} + 2m_0 + 2\frac{I_0}{r_0^2}\,, \quad m_{12} = m_{21} = m_\mathrm{p} l\cos\varphi\,, \quad m_{22} = I_\mathrm{py} + m_\mathrm{p} l^2\,, \tag{3.7}$$

$$m_{33} = I_\mathrm{pz} + 2I_{0xy} + \left(m_0 + \frac{I_0}{r_0^2}\right)\frac{w^2}{2} - (I_\mathrm{pz} - I_\mathrm{px} - m_\mathrm{p} l^2)\sin^2\varphi\,, \tag{3.8}$$

$$d_{12} = -m_\mathrm{p} l\dot{\varphi}\sin\varphi - \frac{2b}{r_0}\,,\ d_{13} = m_\mathrm{p} l\dot{\psi}\sin\varphi\,,\ d_{23} = (I_\mathrm{pz} - I_\mathrm{px} - m_\mathrm{p} l^2)\dot{\psi}\sin\varphi\cos\varphi\,, \tag{3.9}$$

$$d_{31} = m_\mathrm{p} l\dot{\psi}\sin\varphi\,, \quad d_{32} = -d_{23}\,, \quad d_{33} = -(I_\mathrm{pz} - I_\mathrm{px} - m_\mathrm{p} l^2)\dot{\varphi}\sin\varphi\cos\varphi + \frac{w^2}{2r_0^2}b\,, \tag{3.10}$$

where I_{0xy} is the moment of inertia of the wheel in the vertical direction, m_p, I_px, I_py, I_pz, are mass and moments of inertia of the whole pendulum without wheels, l is the length of the rod, w is the distance between the wheels, x is the forward distance traveled by the pendulum, φ is the pitch angle, ψ is the yaw angle and $u_{0\mathrm{L}}$, $u_{0\mathrm{R}}$ are the left and right wheel torques, respectively. Parameters m_0, r_0, I_0, b, g are the same as in the planar robot model.

Next, the planar model and the equilibrium pose from the previous section are used to compute the pendulum's parameters. I obtained the pendulum mass as a sum of the masses of all robot parts except for the wheels:

$$m_\mathrm{p} = 2m_1 + 2m_2 + 2m_3 + 2m_4\,. \tag{3.11}$$

The rod length l was computed as the distance of the wheel and body in the equilibrium position:

$$l = \sqrt{(\boldsymbol{x}_{4,\mathrm{eq}} - \boldsymbol{x}_{0,\mathrm{eq}})^\mathsf{T}(\boldsymbol{x}_{4,\mathrm{eq}} - \boldsymbol{x}_{0,\mathrm{eq}})}\,. \tag{3.12}$$

I decided to neglect the moments of inertia of the links because their values are about two orders of magnitude smaller than the body values. Thus, I set the pendulum inertia to match the body parameters in the Gazebo simulator. The distance w was also selected according to the 3D Gazebo model.

3.1.3 Linearization and Discretization

Next, I continued with the linearization of the 3D wheeled inverted pendulum model (3.6)

$$\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q},\boldsymbol{u})\,, \quad \boldsymbol{q} = \begin{bmatrix}\dot{x} & \dot{\varphi} & \dot{\psi} & x & \varphi & \psi\end{bmatrix}^\mathsf{T}\,, \quad \boldsymbol{u} = \begin{bmatrix}u_{0\mathrm{L}} & u_{0\mathrm{R}}\end{bmatrix}^\mathsf{T}\,, \tag{3.13}$$

by computing the state-space matrices of the linearized continuous system $\mathbf{A}_\mathrm{c}$, $\mathbf{B}_\mathrm{c}$:

$$\Delta\dot{\boldsymbol{q}} = \mathbf{A}_\mathrm{c}\Delta\boldsymbol{q} + \mathbf{B}_\mathrm{c}\Delta\boldsymbol{u}\,, \quad \Delta\boldsymbol{q} = \boldsymbol{q} - \boldsymbol{q}_\mathrm{eq}\,, \quad \Delta\boldsymbol{u} = \boldsymbol{u} - \boldsymbol{u}_\mathrm{eq}\,, \quad \boldsymbol{q}_\mathrm{eq} = \boldsymbol{0}\,, \quad \boldsymbol{u}_\mathrm{eq} = \boldsymbol{0}\,, \tag{3.14}$$

$$\mathbf{A}_\mathrm{c} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}\bigg|_{\boldsymbol{q}_\mathrm{eq},\boldsymbol{u}_\mathrm{eq}}\,, \quad \mathbf{B}_\mathrm{c} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\bigg|_{\boldsymbol{q}_\mathrm{eq},\boldsymbol{u}_\mathrm{eq}}\,. \tag{3.15}$$

From the linearized continuous system, I obtained the discrete-time system with state-space matrices $\mathbf{A}_\mathrm{d}$, $\mathbf{B}_\mathrm{d}$ using *zero-order hold*:

$$\Delta\boldsymbol{q}_{k+1} = \mathbf{A}_\mathrm{d}\Delta\boldsymbol{q}_k + \mathbf{B}_\mathrm{d}\boldsymbol{u}_k\,, \tag{3.16}$$

$$\mathbf{A}_\mathrm{d} = \mathrm{e}^{\mathbf{A}_\mathrm{c}T} \doteq \begin{bmatrix} 0.99212 & 0.00052 & 0 & 0 & -0.05935 & 0 \\ 0.02699 & 0.99823 & 0 & 0 & 0.22980 & 0 \\ 0 & 0 & 0.99590 & 0 & 0 & 0 \\ 0.00100 & 0 & 0 & 1.00000 & -0.00003 & 0 \\ 0.00001 & 0.00100 & 0 & 0 & 1.00012 & 0 \\ 0 & 0 & 0.00100 & 0 & 0 & 1.00000 \end{bmatrix}\,, \tag{3.17}$$

$$\mathbf{B}_\mathrm{d} = \int_0^T \mathrm{e}^{\mathbf{A}_\mathrm{c}t}\mathrm{d}t\,\mathbf{B}_\mathrm{c} \doteq \begin{bmatrix} -0.02757 & -0.02757 \\ 0.09447 & 0.09447 \\ 0.08965 & -0.08965 \\ -0.00001 & -0.00001 \\ 0.00005 & 0.00005 \\ 0.00004 & -0.00004 \end{bmatrix}\,, \tag{3.18}$$

where $T = 0.001$ s is the expected sampling period.

3.1.4 Controller Design

Having the linearized model, I proceeded with the controller design. First, I reduced the model by removing the yaw angle and the forward distance. None of these states is important for stabilizing or steering and the rest of the states are independent of them. Therefore, the reduction was only a matter of removing 4th and 6th rows and columns from $\mathbf{A}_\mathrm{d}$ and 4th and 6th rows from $\mathbf{B}_\mathrm{d}$.

Upon obtaining the state-space matrices $\mathbf{A}_\mathrm{r}$ and $\mathbf{B}_\mathrm{r}$ of the reduced model, I created an augmented system to include integral action [10] in the LQR for speed and yaw rate reference tracking

$$\begin{bmatrix} \Delta\boldsymbol{q}_{\mathrm{r},k+1} \\ \boldsymbol{\varepsilon}_{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{A}_\mathrm{r} & \mathbf{0} \\ \mathbf{L} & \mathbf{I} \end{bmatrix}}_{\mathbf{A}_\mathrm{int}} \begin{bmatrix} \Delta\boldsymbol{q}_{\mathrm{r},k+1} \\ \boldsymbol{\varepsilon}_{k} \end{bmatrix} + \underbrace{\begin{bmatrix} \mathbf{B}_\mathrm{r} \\ \mathbf{0} \end{bmatrix}}_{\mathbf{B}_\mathrm{int}} \boldsymbol{u}_k + \begin{bmatrix} 0 \\ \mathbf{r} \end{bmatrix}, \tag{3.19}$$

$$\mathbf{L} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}, \quad \mathbf{r} = \begin{bmatrix} \dot{x}_\mathrm{ref} \\ \dot{\psi}_\mathrm{ref} \end{bmatrix} \quad \Delta\boldsymbol{q}_\mathrm{r} = \begin{bmatrix} \Delta\dot{x} & \Delta\dot{\varphi} & \Delta\dot{\psi} & \Delta\varphi \end{bmatrix}^\mathsf{T}, \tag{3.20}$$

where $\boldsymbol{\varepsilon}$ is the integrated tracking error, $\mathbf{I}$ is a 2×2 identity matrix, and $\mathbf{r}$ is the reference vector. Then the standard infinite horizon LQR design procedure [11] followed to obtain the state feedback gain $\mathbf{K}$ for the wheel stabilization control law with state and input weight matrices $\mathbf{Q}_\mathrm{LQR}$, $\mathbf{R}_\mathrm{LQR}$:

$$\begin{bmatrix} u_{0L} & u_{0R} \end{bmatrix}^\mathsf{T} = -\mathbf{K} \begin{bmatrix} \Delta\boldsymbol{q}_\mathrm{r} & \boldsymbol{\varepsilon} \end{bmatrix}^\mathsf{T}, \quad \mathbf{R}_\mathrm{LQR} = \begin{bmatrix} 1000.000 & 0.000 \\ 0.000 & 1000.000 \end{bmatrix}, \tag{3.21}$$

$$\mathbf{Q}_\mathrm{LQR} = \begin{bmatrix} 4081.633 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 1.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 313.470 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 1.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.082 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.002 \end{bmatrix}, \tag{3.22}$$

with the diagram of the feedback loop shown in Figure 3.2.

The mapping of the pendulum states to the robot states that are measured (or estimated) is following:

$$\Delta\dot{x} = \frac{r_0}{2}(\dot{\theta}_\mathrm{0R} + \dot{\theta}_\mathrm{0L}), \quad \Delta\dot{\varphi} = \dot{\theta}_1 + \dot{\theta}_2 + \dot{\theta}_3, \quad \Delta\dot{\psi} = \frac{r_0}{w}(\dot{\theta}_\mathrm{0R} - \dot{\theta}_\mathrm{0L}), \tag{3.23}$$

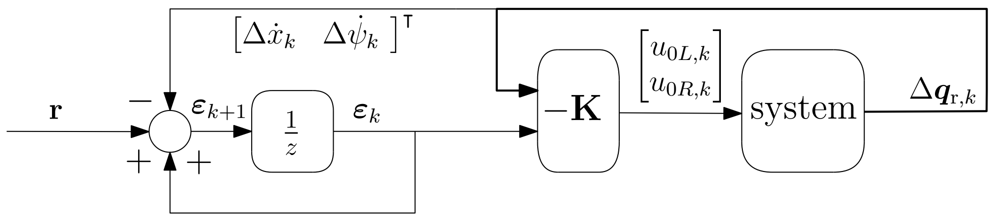

$[\Delta\dot{x}_k \quad \Delta\dot{\psi}_k]^\mathsf{T}$
$\mathbf{r}$
$-$
$+$
$+$
$\boldsymbol{\varepsilon}_{k+1}$
$\frac{1}{z}$
$\boldsymbol{\varepsilon}_k$
$-\mathbf{K}$
$\begin{bmatrix} u_{0L,k} \\ u_{0R,k} \end{bmatrix}$
system
$\Delta\boldsymbol{q}_{\mathrm{r},k}$

Figure 3.2: Control loop of the proposed LQR.

$$\Delta\varphi = \theta_1 + \theta_2 + \theta_4 - \theta_{1,\mathrm{eq}} - \theta_{2,\mathrm{eq}} - \theta_{4,\mathrm{eq}} \,, \tag{3.24}$$

where the subscripts R and L denote the right or left side of the robot.

The body motors are controlled using simple PD control law:

$$u_{4\mathrm{R,L}} = -p(\Delta\theta_{4\mathrm{R,L}} - \Delta\theta_{4,\mathrm{ref}}) - d\dot{\theta}_4 \,, \tag{3.25}$$

$$\Delta\theta_{4\mathrm{R,L}} = \theta_{4\mathrm{R,L}} - \theta_{4,\mathrm{eq}} \,, \quad \Delta\theta_{4,\mathrm{ref}} = \theta_{4,\mathrm{ref}} - \theta_{4,\mathrm{eq}} \,, \tag{3.26}$$

with p, d values tuned experimentally. The reference is used to change the length of robot's legs with default value $\Delta\theta_{4,\mathrm{ref}} = 0$.

3.1.5 Balancing Performance

I examined the balancing performance of the controller for three different values of leg lengths. The first one is the default length given by the equilibrium depicted in Figure 2.2. The other two cases correspond to stretched and squatted robot poses shown in Figure 3.3 with $\Delta\theta_{4,\mathrm{ref}} = 0.15$ rad and $\Delta\theta_{3,\mathrm{ref}} = -0.15$ rad, respectively.

In the test scenario, I first applied a 0.2 ms long force impulse of 2 kN to the back of the robot in rest, simulating a significant push in the forward direction. After two seconds, I applied the same force impulse to the robot's side to see how well the body controllers perform.

The responses of the system varied with the Gazebo simulator settings, in particular with the Constraint Force Mixing[1] (CFM) parameter value. This parameter is used to increase simulation stability by converting hard joint constraints to soft constraints, reducing accuracy. With the CFM set to a low value or zero, spikes in joint angles and velocities occur during the simulation. On the other hand, with high CFM, simulation behavior is unnatural. I

[1] http://www.ode.org/ode-latest-userguide.html#sec_3_8_0

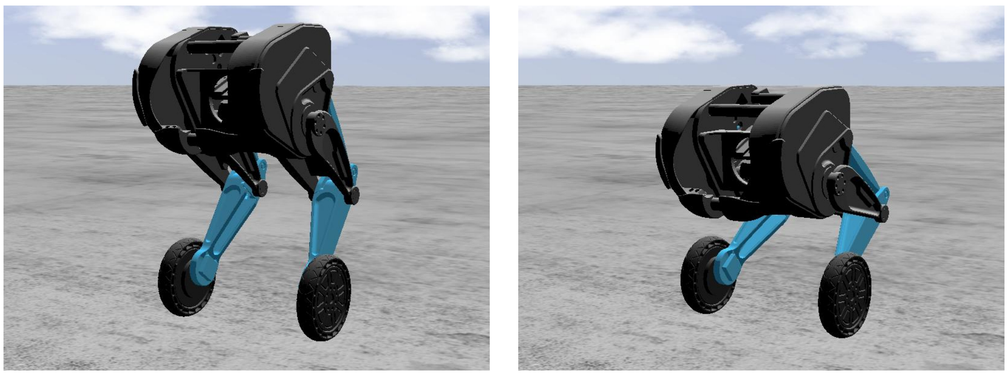


Figure 3.3: Stretched and squatted balancing pose.

always tried to set CFM as low as possible. In most simulations, that was between 10^{-3} and 10^{-4}.

The robot in the default pose handled both disturbances well which is shown in Figures 3.4 and 3.5. The peak torque values for the back push were around -0.7 Nm for both u_0 and 0.4 Nm for u_4. After the side hit, u_4 peaked at ± 0.5 Nm, and almost no torque was applied on the wheels. After the side push, the visible oscillations are caused by the weight load transfer between the robot's legs.

The stretched robot performed similarly as in the default pose which is reflected in Figures 3.6 and 3.7. The only exception is a u_4 spike right after the back push, which might be caused by the numerical error related to the CFM parameter. The non-zero steady-state value of u_4 and $\Delta\varphi$ is just a consequence of stabilization in the non-equilibrium pose. For the back push, u_0 again peaked at around -0.7 Nm and u_4 at ± 0.4 Nm difference from the steady-state value (if the spike is not counted). The side push behavior was also comparable with u_4 peak at ± 0.5 Nm steady-state difference and negligible wheel torques.

The back push response for the squatted pose again matched the default pose behavior, as shown in Figure 3.8, with a slightly higher wheel torque peak of -0.8 Nm. The side push behavior captured in Figure 3.9 corresponds to the other two poses with the same u_4 peak at ± 0.5 Nm steady-state difference.

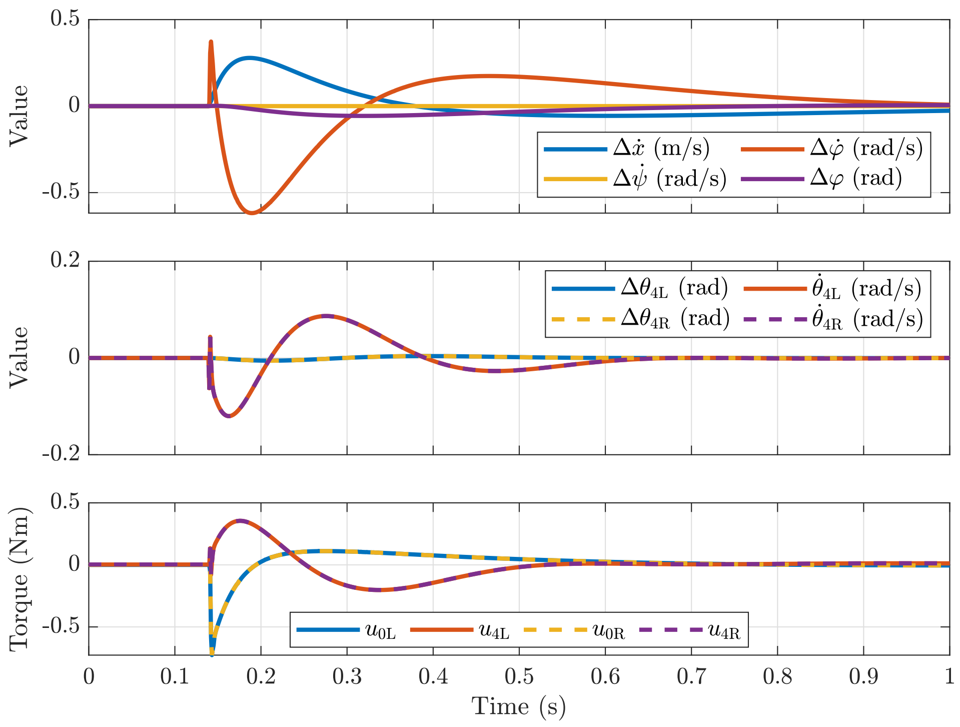


Figure 3.4: Default pose — back push response.

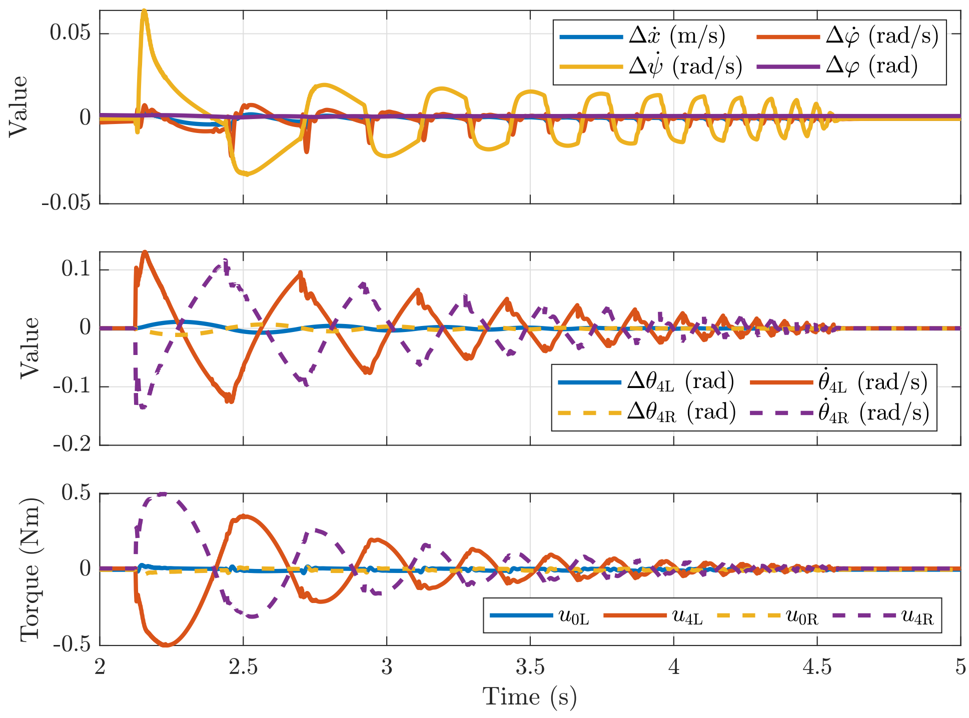


Figure 3.5: Default pose — side push response.

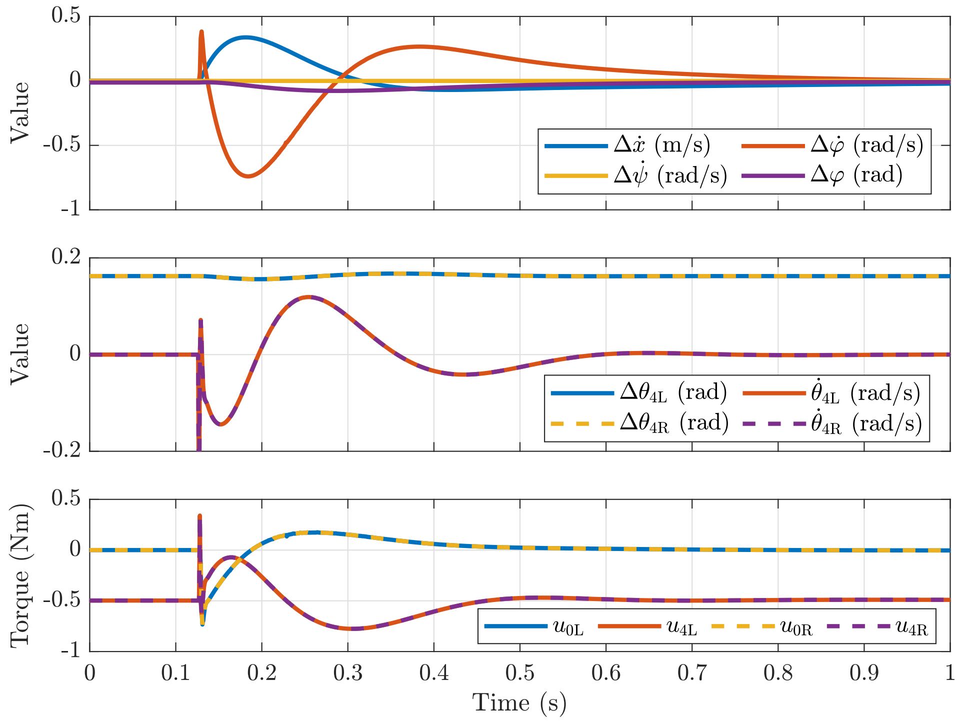


Figure 3.6: Stretched pose — back push response.

3.1.6 Steering Performance

I tested the steering performance for the three poses with a simple reference trajectory. First, the robot was required to reach a velocity of 0.5 m/s. Then, it should start turning left with 1.1 rad/s yaw rate while keeping the velocity. After 3 seconds of turning, the robot was supposed to stop moving.

In all the cases, the robot was able to follow the references without a problem as can be seen from Figures 3.10, 3.11 and 3.12. It reaches the speed and yaw reference (denoted by blue and yellow dashed lines in the top graphs) after 1 s with peak wheel torque values around -0.2 Nm. In the default and stretched pose, a load transfer between the legs occurs during the cornering, possibly due to the centrifugal force. When the turning stops, the load transfer leads to similar oscillatory behavior as in the side push balancing test. The centrifugal force effect in the squatted pose is probably negligible, and thus the oscillations are not observed.

Despite the simplicity of the selected approach, the controller can stabilize and steer the robot even with varying leg lengths. This was also confirmed in experiments with the real robot described in Chapter 4.

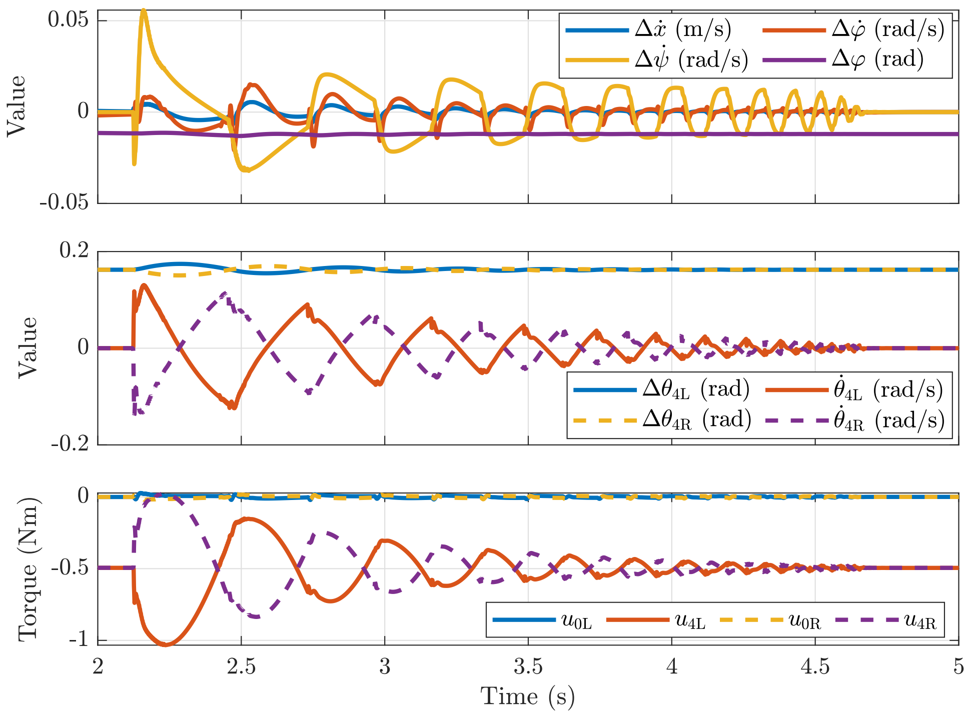


Figure 3.7: Stretched pose — side push response.

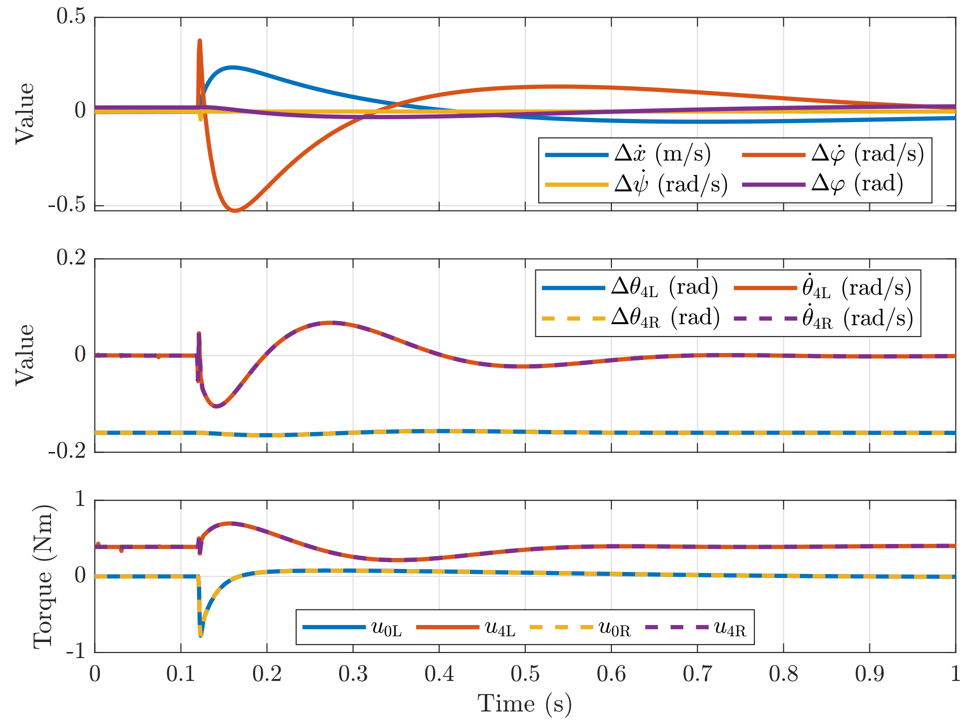


Figure 3.8: Squatted pose — back push response.

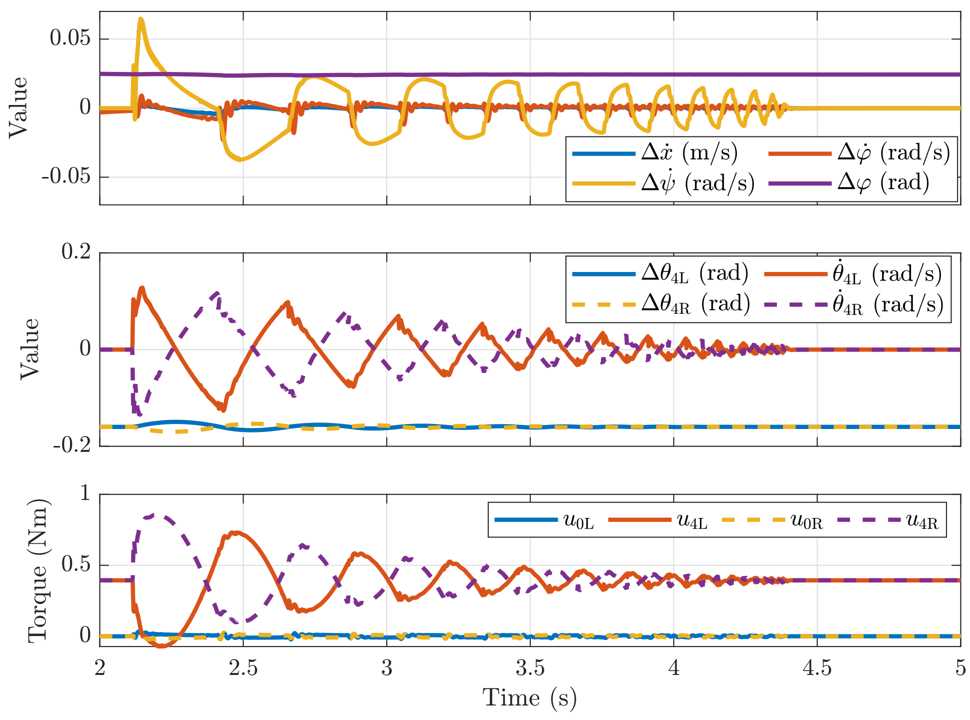


Figure 3.9: Squatted pose — side push response.

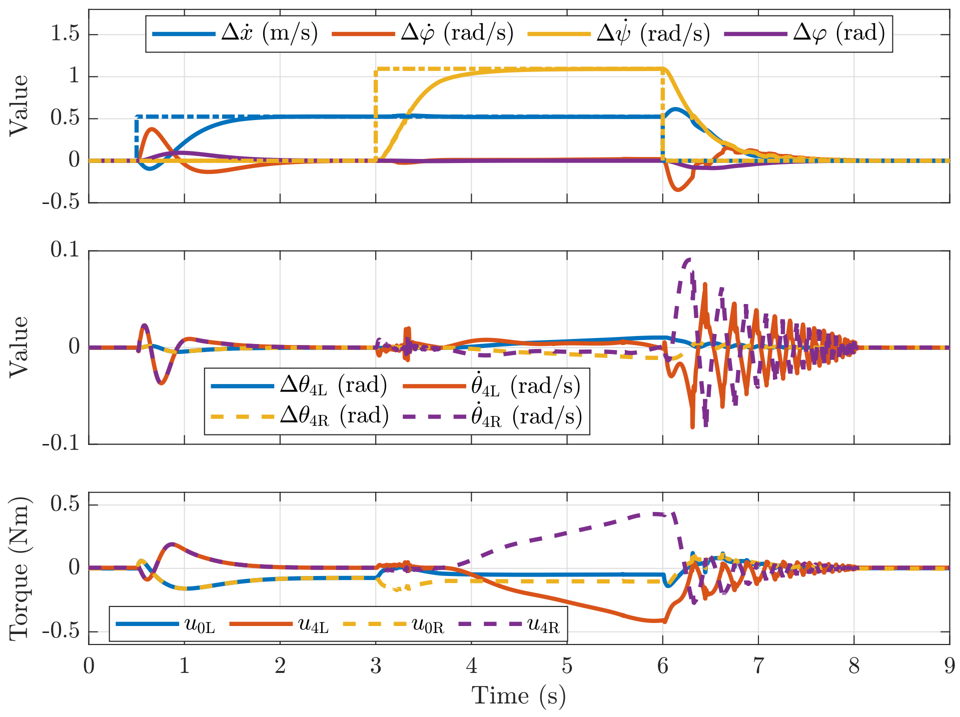


Figure 3.10: Default pose — steering.

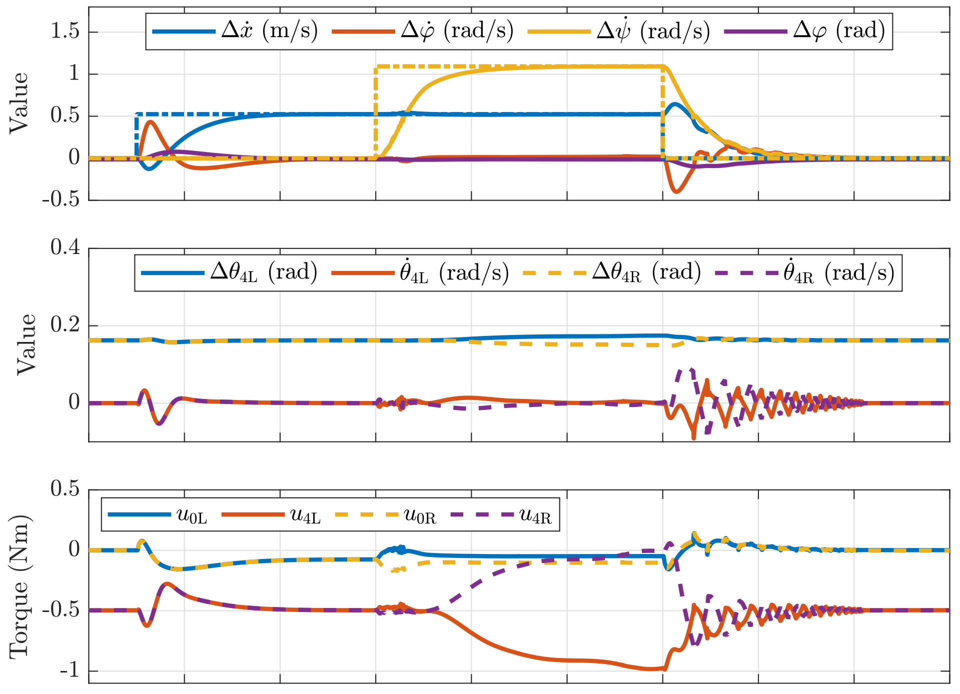


Figure 3.11: Stretched pose — steering.

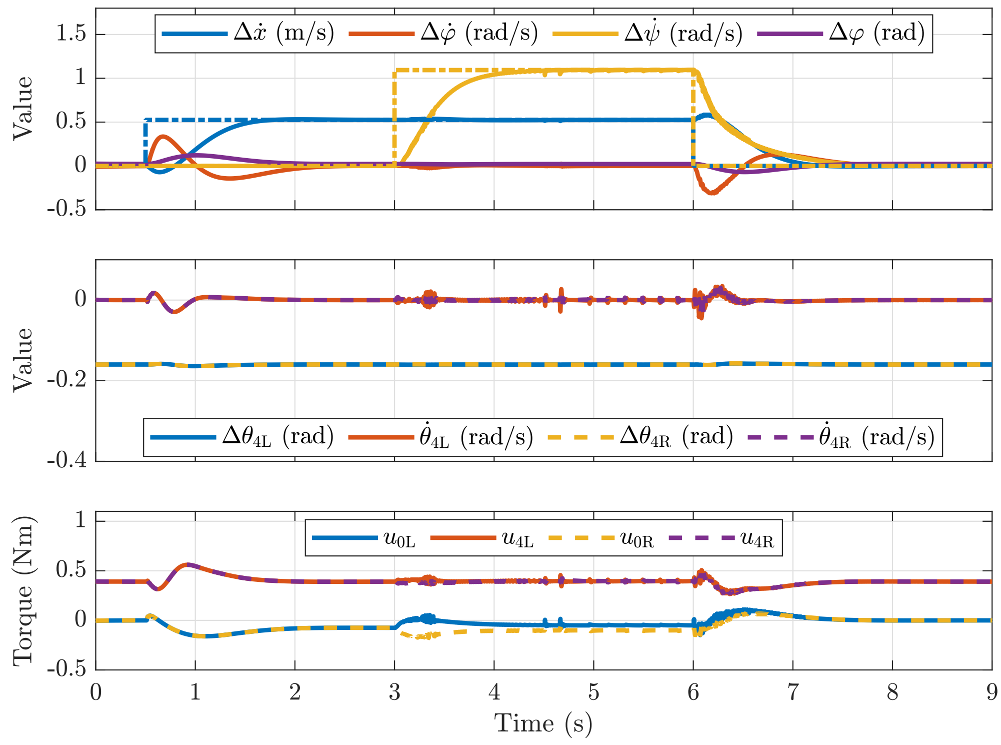


Figure 3.12: Squatted pose — steering.

3.2 State Observer

As I mentioned in the previous section, I originally intended to use two state feedback controllers based on the planar model. Only one of the angles (θ_4) and angular velocities ($\dot{\theta}_4$) in the closed chain of the leg are measured and only sums of θ_0, θ_1 and $\dot{\theta}_0$, $\dot{\theta}_1$ are available from the encoders. Thus, they had to be computed or estimated. For this purpose, I implemented the Unscented Kalman Filter, which, apart from the computation of unmeasured quantities, filters, and fuses all the available measurements. This is convenient even for the pendulum state feedback that has the necessary data at its disposal.

3.2.1 Unscented Kalman Filter

The Unscented Kalman Filter (UKF) [12],[13] is a Kalman filter variant that uses so-called *unscented transformation* to estimate mean and covariance of the state distribution propagated through the non-linear dynamics and measurements of the system. The UKF is more accurate [12],[13] than the Extended Kalman Filter (which relies on system linearization for the distribution estimation) with the same order of computational complexity.

In the unscented transformation, a set of points called *sigma points* is chosen such that its sample mean and sample covariance is the same as for the state distribution. Then, the non-linear transformation is applied to each point separately. Their new weighted sample mean and sample covariance are used for the estimation of the new distribution.

I implemented the UKF with *nonagumented* unscented transform [14] assuming only white additive Gaussian process and measurement noise with covariances $\mathbf{Q}$, $\mathbf{R}$. In the data update step, the $2n+1$ sigma points $\boldsymbol{s}_i^{\sigma+}$ and their weights W_i are computed from the previous predicted value $\hat{\boldsymbol{s}}_{k-1}^{+}$ and its covariance $\boldsymbol{P}_{k-1}^{+}$:

$$\boldsymbol{s}_0^{\sigma+} = \hat{\boldsymbol{s}}_{k-1}^{+}\,, \quad W_0 = \frac{\kappa}{n+\kappa}\,, \tag{3.27}$$

$$\boldsymbol{s}_i^{\sigma+} = \hat{\boldsymbol{s}}_{k-1}^{+} + \left(\sqrt{(n+\kappa)\boldsymbol{P}_{k-1}^{+}}\right)_i\,, \quad W_i = \frac{1}{2(n+\kappa)}\,, \quad i = 1, \dots, n\,, \tag{3.28}$$

$$\boldsymbol{s}_i^{\sigma+} = \hat{\boldsymbol{s}}_{k-1}^{+} - \left(\sqrt{(n+\kappa)\boldsymbol{P}_{k-1}^{+}}\right)_i\,, \quad W_i = \frac{1}{2(n+\kappa)}\,, \quad i = n+1, \dots, 2n+1\,, \tag{3.29}$$

where n in the number of the states, κ is a scaling parameter and $\left(\sqrt{(n+\kappa)\boldsymbol{P}_{k-1}^{+}}\right)_i$ is the ith column of matrix square root of $(n+\kappa)\boldsymbol{P}_{k-1}^{+}$ (that may be obtained from Cholesky factorization if the matrix is positive definite). The sigma points are transformed by the output function $\boldsymbol{h}(\boldsymbol{s})$, compared with the actual measurement $\boldsymbol{y}$ and the state estimate is updated:

$$\boldsymbol{y}_i^{\sigma} = \boldsymbol{h}(\boldsymbol{s}_i^{\sigma+})\,, \quad \boldsymbol{y}^{\sigma} = \sum_{i=0}^{2n+1} W_i \boldsymbol{y}_i^{\sigma}\,, \tag{3.30}$$

$$\boldsymbol{P}_y = \sum_{i=0}^{2n+1} W_i \left(\boldsymbol{y}_i^{\sigma} - \boldsymbol{y}^{\sigma}\right)\left(\boldsymbol{y}_i^{\sigma} - \boldsymbol{y}^{\sigma}\right)^{\mathsf{T}} + \mathbf{R}\,, \tag{3.31}$$

$$\boldsymbol{P}_{sy} = \sum_{i=0}^{2n+1} W_i \left(\boldsymbol{s}_i^{\sigma+} - \hat{\boldsymbol{s}}_{k-1}^{+}\right)\left(\boldsymbol{y}_i^{\sigma} - \boldsymbol{y}^{\sigma}\right)^{\mathsf{T}}\,, \tag{3.32}$$

$$\boldsymbol{P}_k^{-} = \boldsymbol{P}_{k-1}^{+} - \boldsymbol{P}_{sy}\boldsymbol{P}_y^{-1}\boldsymbol{P}_{sy}^{\mathsf{T}}\,, \quad \hat{\boldsymbol{s}}_k^{-} = \hat{\boldsymbol{s}}_{k-1}^{+} + \boldsymbol{P}_{sy}\boldsymbol{P}_y^{-1}\left(\boldsymbol{y} - \boldsymbol{y}^{\sigma}\right)\,. \tag{3.33}$$

In the time update step, the sigma points $\boldsymbol{s}_i^{\sigma-}$ are computed from $\hat{\boldsymbol{s}}_k^{-}$ and $\boldsymbol{P}_k^{-}$ analogously as in the data update step (3.27), (3.28), (3.29). Then, they are transformed applying the discrete (or discretized) system dynamics $\boldsymbol{z}(\boldsymbol{s}, \boldsymbol{u})$ with inputs $\boldsymbol{u}$ and the states are predicted:

$$\tilde{\boldsymbol{s}}_i^{\sigma} = \boldsymbol{z}(\boldsymbol{s}_i^{\sigma-}, \boldsymbol{u})\,, \quad \tilde{\boldsymbol{s}}^{\sigma} = \sum_{i=0}^{2n+1} W_i \tilde{\boldsymbol{s}}_i^{\sigma}\,, \tag{3.34}$$

$$\boldsymbol{P}_k^{+} = \sum_{i=0}^{2n+1} W_i \left(\tilde{\boldsymbol{s}}_i^{\sigma} - \tilde{\boldsymbol{s}}^{\sigma}\right)\left(\tilde{\boldsymbol{s}}_i^{\sigma} - \tilde{\boldsymbol{s}}^{\sigma}\right)^{\mathsf{T}} + \mathbf{Q}\,, \quad \hat{\boldsymbol{s}}_k^{+} = \tilde{\boldsymbol{s}}^{\sigma}\,. \tag{3.35}$$

Such implementation does not enforce the angle constraints $\boldsymbol{a}(\boldsymbol{s})$ (2.15) because they are included in the model only through the pseudovelocities. Hence, their values would not be estimated correctly. I overcame this by introducing the constraints as *perfect measurements* [15], which is done by expanding the measurement transformation:

$$\boldsymbol{h}_{\mathrm{e}}(\boldsymbol{s}) = \begin{bmatrix} \boldsymbol{h}(\boldsymbol{s}) \\ \boldsymbol{a}(\boldsymbol{s}) \end{bmatrix}, \quad \boldsymbol{y}_{\mathrm{e}} = \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{0} \end{bmatrix}, \quad \mathbf{R}_{\mathrm{e}} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{3.36}$$

I implemented two separate UKFs for each side of the robot with shared pitch and pitch rate data. The available roll (roll rate) and yaw (yaw rate) information were not used because it would require coupling between the sides, which is not included in the simplified planar model. The measurement noise covariance was selected based on the worst-case expectation for the real robot measurements (the real robot had not been constructed yet at the time of the design).

3.2.2 Estimation and Control Performance

I tested the estimation and control performance with the observer included in the feedback loop in the same two scenarios as before (see subsections 3.1.5 and 3.1.6). The estimation performance for all three leg lengths was similar, which is not surprising since UKF is a non-linear observer. Thus, I will present the results only for the default pose.

In the balancing test, the robot managed to withstand back, and side pushes as depicted in Figures 3.13 and 3.15. The response was slightly slower than in the ideal case, but this was to be expected. Otherwise, the performance was decent, and the state estimates closely matched the real values, which is reflected in Figures 3.14 and 3.16.

The robot could also follow the references in the steering test as can be seen from Figure 3.17, yet the oscillatory behavior after the cornering is much worse. Figure 3.18 shows, that the $\dot{\theta}_1$ and $\dot{\theta}_3$ estimates drifted from the real values while the robot was turning left. After the turning was over, the estimates shifted back while heavily oscillating. This, again, is caused by the fact that the coupling and load transfer between the two sides while turning is not considered.

Overall, the state estimation worked well for balancing and forward (backward) movement. The control system was also able to steer the robot, even though the estimates shifted during the turn. The coupling of the two robot sides should be introduced to improve the estimation, ideally, by using the full robot model for the UKF design instead of two independent planar models.

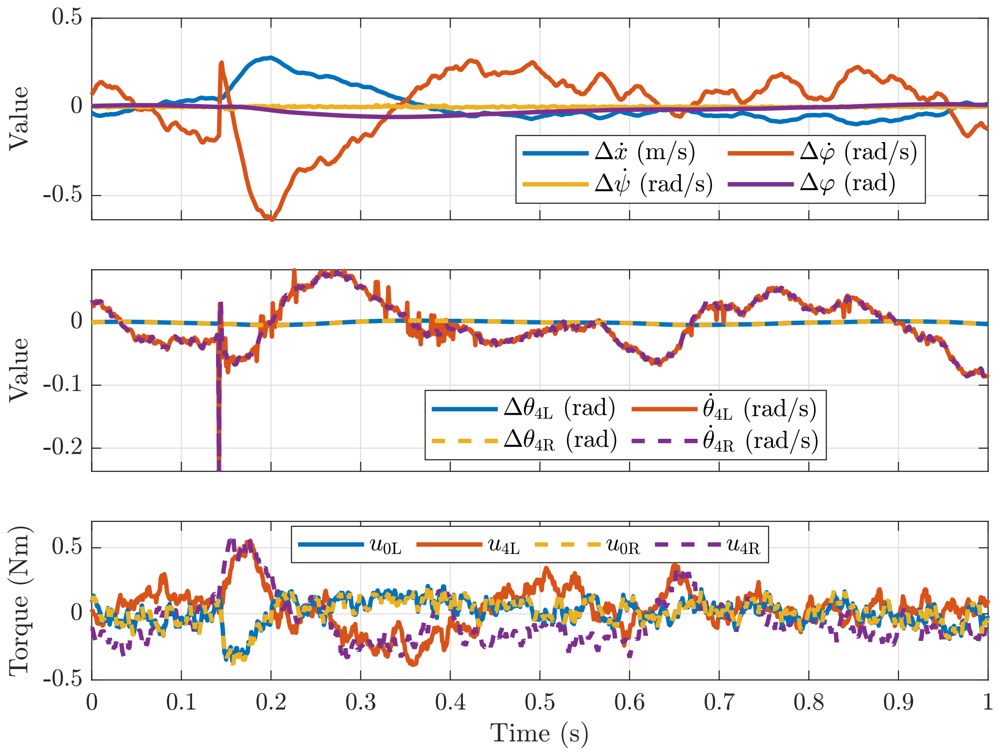


Figure 3.13: Back push response with UKF.

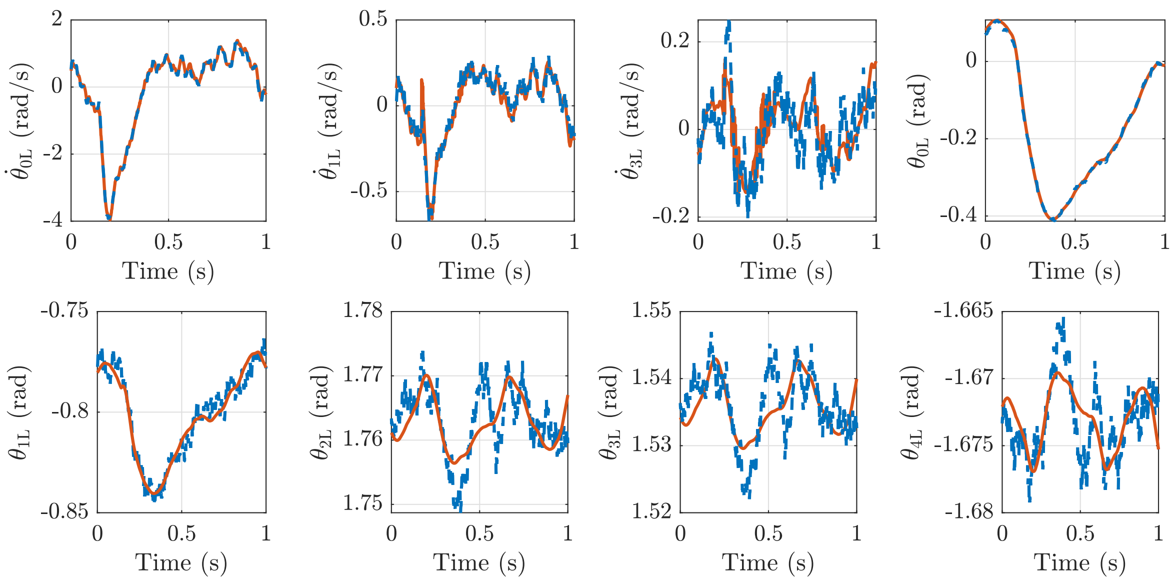


Figure 3.14: Back push — left side comparison of true (red) and estimated values (blue).

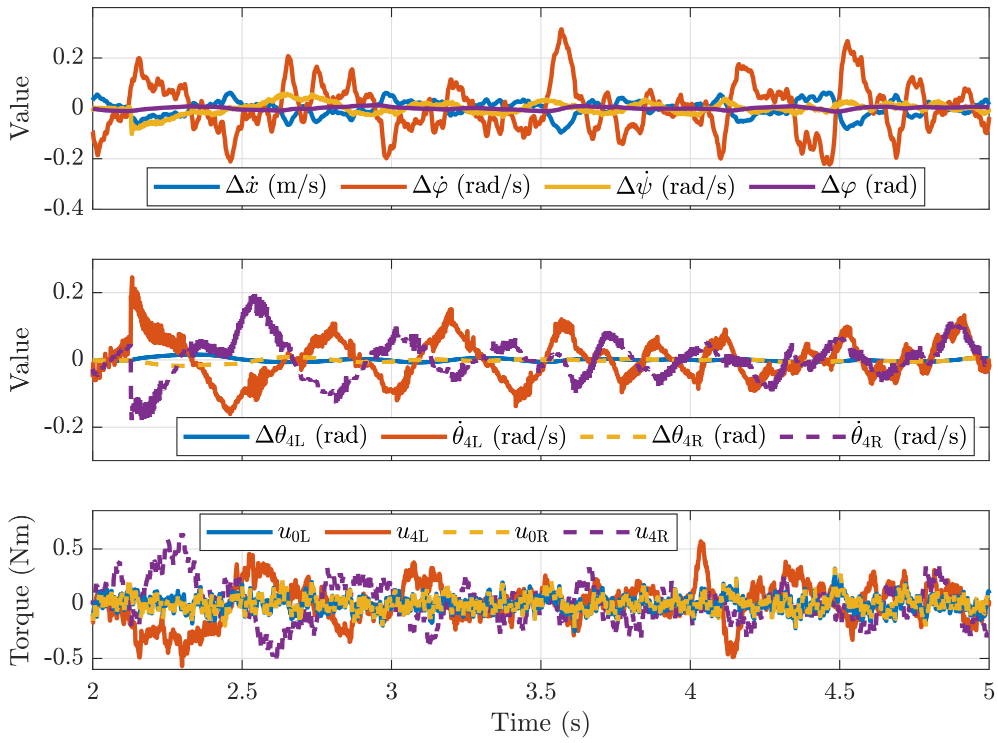


Figure 3.15: Side push response with UKF.

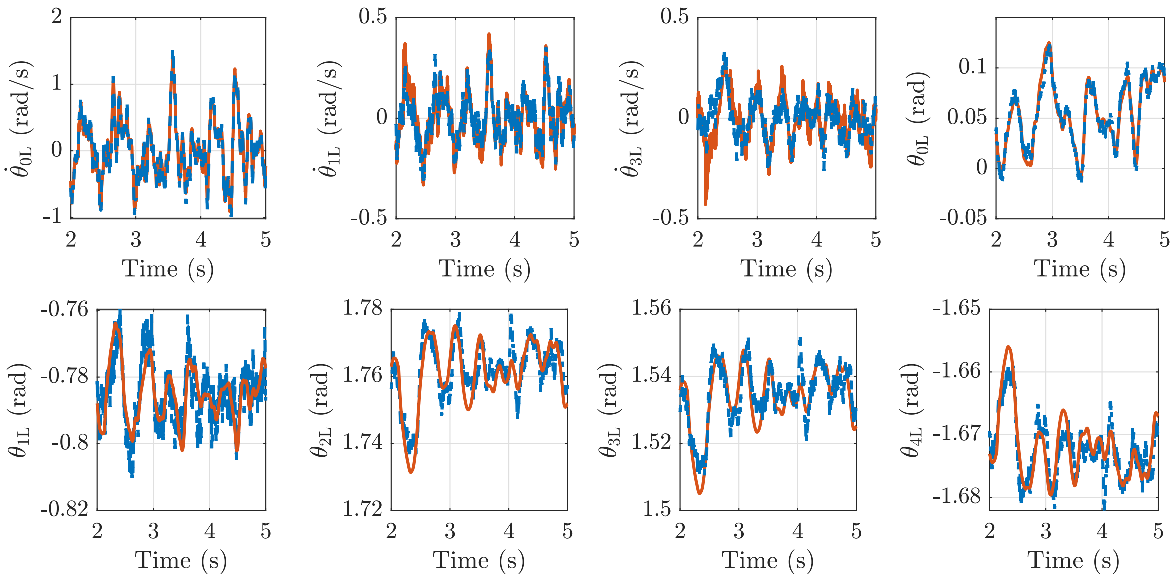


Figure 3.16: Side push — left side comparison of true (red) and estimated values (blue).

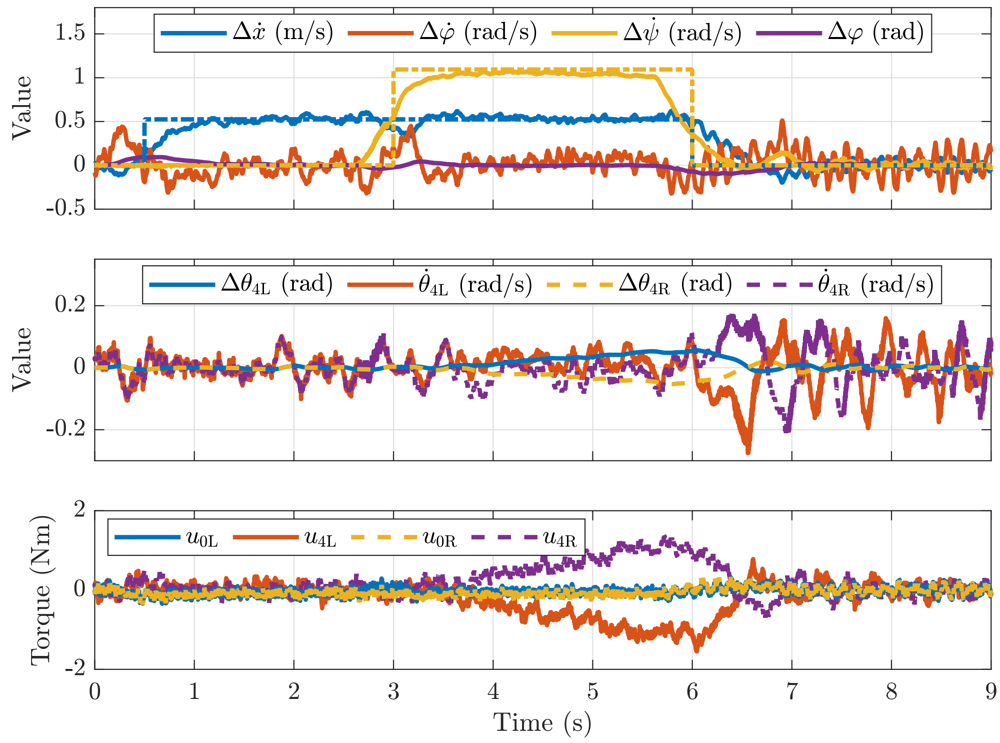


Figure 3.17: Steering performance with UKF.

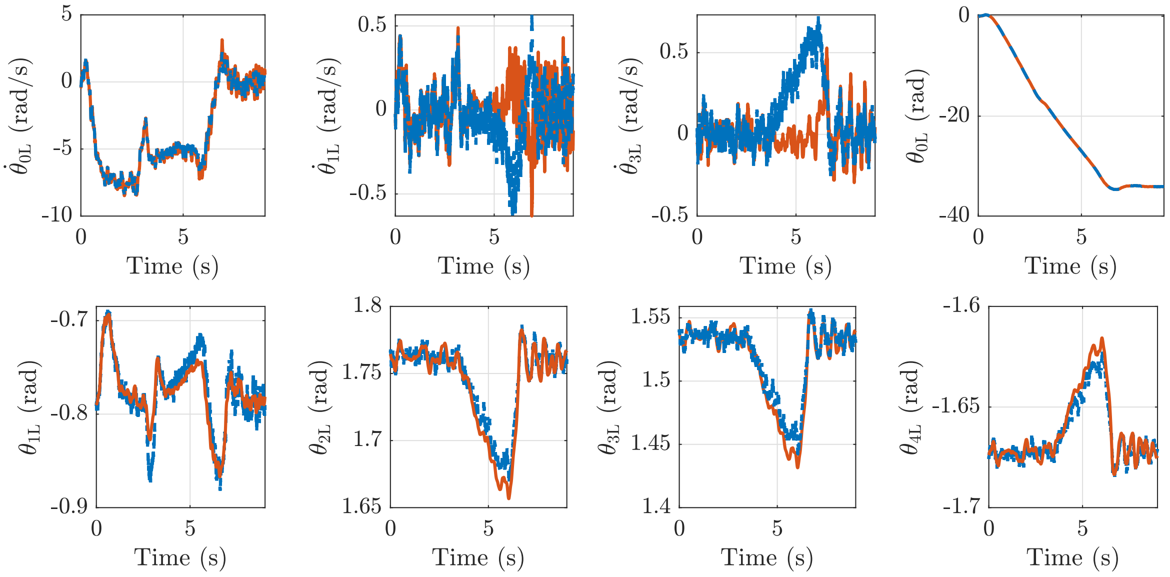


Figure 3.18: Steering — left side comparison of true (red) and estimated values (blue).

3.3 Jumping Maneuver

In this section, I describe the jumping maneuver trajectory computation, which is a simplification of the method described in [16]. First, I rederived the planar model of the robot (2.27),(2.31) so that it does not include constant wheel height and no-slip conditions directly but enforce them via Lagrange multipliers representing the ground reaction forces. Then, I used non-linear optimization to find the jumping trajectory, which satisfies the discretized dynamics. I tested this trajectory by applying the obtained control sequence as feedforward in the 3D simulation.

3.3.1 Expanded Model

The derivation of the expanded model is exactly the same as in Chapter 2, the only difference is the addition of two generalized coordinates representing the position of the wheel and two constraints

$$\boldsymbol{x}_0 = \begin{bmatrix} x_0 & y_0 \end{bmatrix}^\mathsf{T}, \quad \bar{\boldsymbol{a}} = \begin{bmatrix} x_0 - r_0\theta_0 & y_0 - r_0 \end{bmatrix}^\mathsf{T} = \boldsymbol{0}\,. \tag{3.37}$$

I skip the derivation to avoid any unnecessary repetition. The resulting model, that corresponds to (2.29) is the following:

$$\underbrace{\bar{\boldsymbol{G}}(\boldsymbol{\theta})^\mathsf{T}\begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{\tau}_{\text{in}} \end{bmatrix}}_{\bar{\mathbf{B}}(\boldsymbol{z})\bar{\boldsymbol{u}}} = \bar{\boldsymbol{M}}(\boldsymbol{z})\dot{\boldsymbol{w}} + \bar{\boldsymbol{C}}(\boldsymbol{z},\boldsymbol{w})\boldsymbol{w} + \underbrace{\bar{\boldsymbol{G}}^\mathsf{T}(\boldsymbol{\theta})\boldsymbol{g}_\text{e}(\boldsymbol{z})}_{\bar{\boldsymbol{g}}(\boldsymbol{z})} - \bar{\boldsymbol{J}}^\mathsf{T}\bar{\boldsymbol{\lambda}}\,, \tag{3.38}$$

$$\bar{\boldsymbol{u}} = \begin{bmatrix} u_0 \\ u_4 \end{bmatrix}, \ \boldsymbol{w} = \begin{bmatrix} \dot{\boldsymbol{x}}_0 \\ \boldsymbol{v} \end{bmatrix}, \ \boldsymbol{z} = \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{\theta} \end{bmatrix}, \ \bar{\boldsymbol{J}} = \frac{\partial \boldsymbol{a}_\text{e}}{\partial \boldsymbol{z}}\bar{\boldsymbol{G}} = \begin{bmatrix} 1 & 0 & -r_0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \ \bar{\boldsymbol{\lambda}} = \begin{bmatrix} \lambda_\text{s} \\ \lambda_\text{g} \end{bmatrix}. \tag{3.39}$$

3.3.2 Optimization

A feasible state trajectory is constrained by integrated dynamics, which I obtained form (3.38) with the backward Euler method similarly as in [16]:

$$\boldsymbol{z}_k - \boldsymbol{z}_{k+1} + T\bar{\boldsymbol{G}}_{k+1}\boldsymbol{w}_{k+1} = \boldsymbol{0}\,, \tag{3.40}$$

$$\bar{\boldsymbol{M}}_{k+1}(\boldsymbol{w}_{k+1} - \boldsymbol{w}_k) + T\left(\bar{\boldsymbol{C}}_{k+1} + \bar{\boldsymbol{g}}_{k+1} - \bar{\mathbf{B}}_{k+1}\bar{\boldsymbol{u}}_{k+1} - \boldsymbol{J}_{k+1}^\mathsf{T}\bar{\boldsymbol{\lambda}}_{k+1}\right)\,, \tag{3.41}$$

with $\bar{\boldsymbol{M}}_{k+1} = \bar{\boldsymbol{M}}(\boldsymbol{z}_k)$ and likewise.

The jump can be divided into two phases. During the first phase, the robot is on the ground preparing to jump. That means that constraints $\bar{\boldsymbol{a}}$ (or equivalently $\bar{\boldsymbol{J}}\boldsymbol{w}$) are active and should be equal to zero. In the second phase, the robot is in the air. Therefore, $\bar{\boldsymbol{\lambda}} = 0$ as no reaction forces from the ground are applied on the wheel. From this, I formulated the following constraints:

$$\bar{\boldsymbol{J}}_k \boldsymbol{w}_k = \boldsymbol{0}\,, \quad \lambda_{\mathrm{g},k} \geq 0\,, \quad k = 1, \ldots, N_{\mathrm{jump}} - 1\,, \tag{3.42}$$

$$\boldsymbol{\lambda}_k = \boldsymbol{0}\,, \quad k = N_{\mathrm{jump}}, \ldots, N\,, \tag{3.43}$$

$$y_{0,N} = r_0\,, \tag{3.44}$$

where N_{jump} is the discrete time of the jump and N is the final time of the whole maneuver. The condition $\lambda_{\mathrm{g},k} \geq 0$ in (3.42) represents the fact, that the reaction force that prevents the wheel from penetrating the ground is non-negative. The constraint (3.44) ensures that the wheel is at ground level in the end of the air phase. More constraints might be used to tweak the jump properties or to limit maximal and minimal state and input values. Also, the landing can be added analogously to the jump preparation phase, if a landing reference is desired. The cost function should be any nonlinear function based on the states, inputs and multipliers. I chose a quadratic cost on inputs to avoid needlessly high inputs:

$$\underset{\bar{\boldsymbol{u}}_1, \ldots, \bar{\boldsymbol{u}}_N}{\text{minimize}} \sum_{k=1}^{N} \bar{\boldsymbol{u}}_k^{\intercal} \bar{Q} \bar{\boldsymbol{u}}_k\,. \tag{3.45}$$

It is worth mentioning that this simplified approach is by no means perfect. It requires manual tuning of N_{jump} and N for given sample time T, which might be made slightly less inconvenient by setting a variable sample time. Appropriate time interval values could be obtained from the resulting trajectory and then adapted for a chosen fixed sample time. Furthermore, an adaptation of the full method suggested in [16] would resolve this problem.

I formulated the problem in Matlab using CasADi [17], which is a framework that allows the formulation of an optimization task using a high-level programming language. CasADi uses algebraic differentiation to compute the necessary derivatives and then calls a general non-linear problem solver, particularly interior point optimizer (IPOPT). That made the implementation both easy and efficient. For N values around 500, the problem converged within a couple of minutes after the program was initialized (on my laptop with Intel i7-8750H CPU).

3.3.3 Results

I tested the method in two test scenarios by applying the input values as a feedforward control sequence in the 3D simulation. The stabilization was switched off during the maneuver and turned on right after it. This is not ideal because such an approach heavily relies on the model's precision during the jump and stabilization control for the landing. In [3], authors improved this by implementing heuristic feedforward control with five different phases and ground contact detection. An even better solution would be the use of the jump trajectory tracking controller. Nevertheless, the simulation experiments still serve as proof that the generated trajectories are valid.

The first test scenario was an upward jump. The optimization result can be seen in Figure 3.19 with an achieved jump height of 20 cm above the ground. Besides limiting the maximal input values, I restricted the wheel height y_0 in the middle of the jump phase to achieve such height. The feedforward 3D simulation trajectory is almost indistinguishable from the precomputed one, as shown in Figure 3.20. The only visible difference in the comparison from Figure 3.21 is that in the 3D simulation, the robot is slightly more tilted.

The second test scenario was a forward jump. Figure 3.22 shows the obtained state trajectory and input signals. The jump height was 3.5 cm, and the jumped distance was around 30 cm. In this case, I also limited the maximal input values and added a jump distance requirement based on the x_0 value at time N. Again, the trajectory from the 3D simulation is nearly identical, which is illustrated in Figure 3.23. The visual comparison in Figure 3.25 confirms that as well. Even though the forward jump trajectories matched the reference closely, the stabilization after the jump was not always successful, especially for higher jumps; this only highlights the drawbacks of the feedforward and stabilization switching approach.

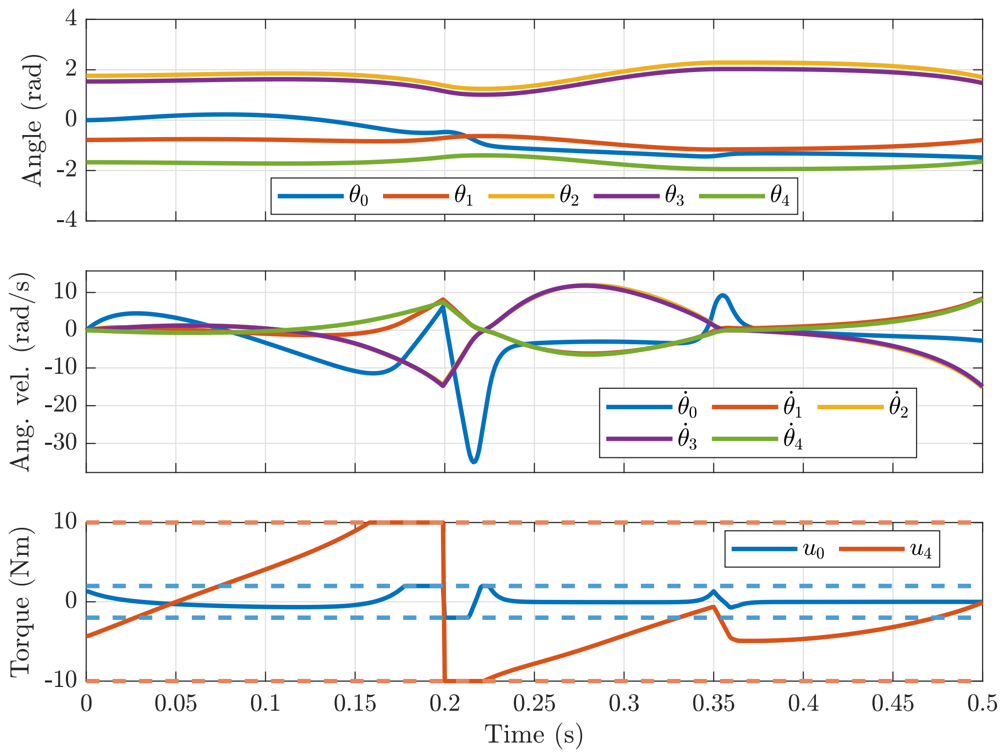


Figure 3.19: Upward jump — state trajectory and input sequence obtained from optimization.

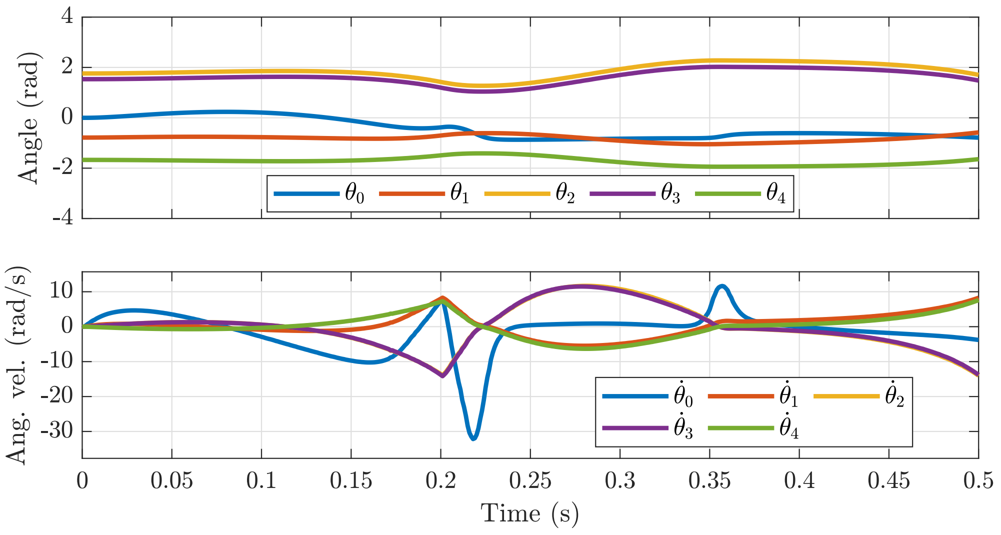


Figure 3.20: Upward jump — feedforward 3D simulation state trajectory.

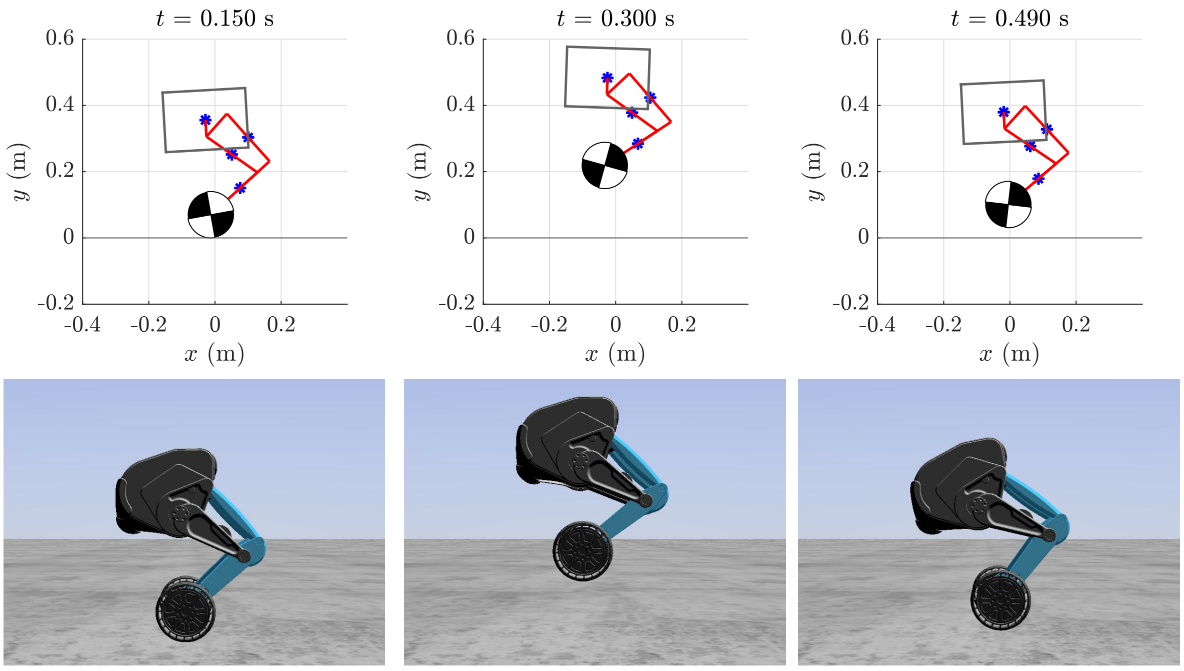


Figure 3.21: Upward jump — visual comparison of simplified model simulation (top) and 3D feedforward simulation (bottom).

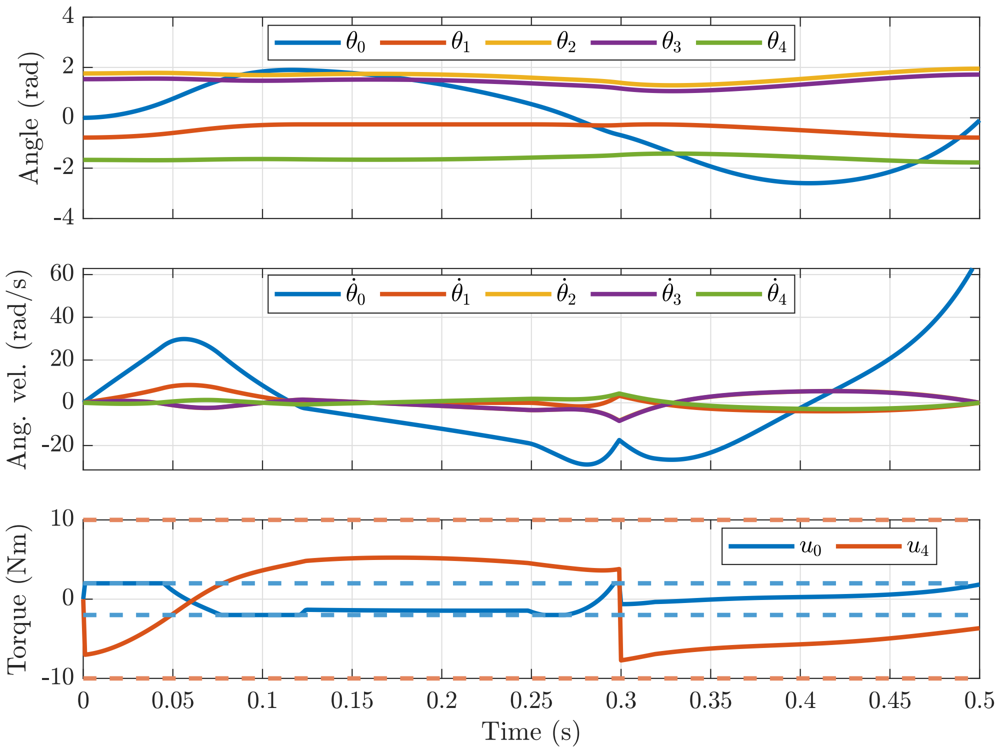


Figure 3.22: Forward jump — state trajectory and input sequence obtained from optimization.

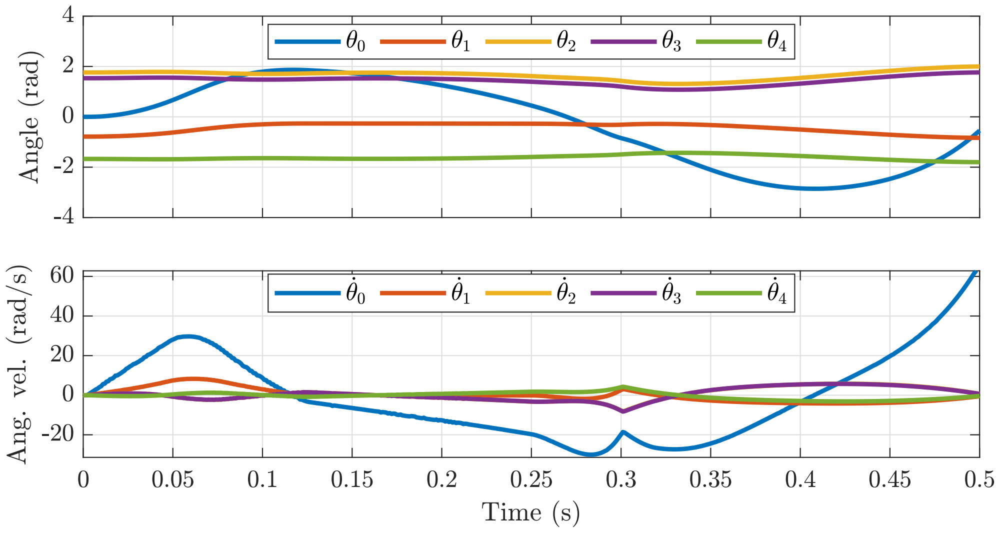


Figure 3.23: Forward jump — feedforward 3D simulation state trajectory.

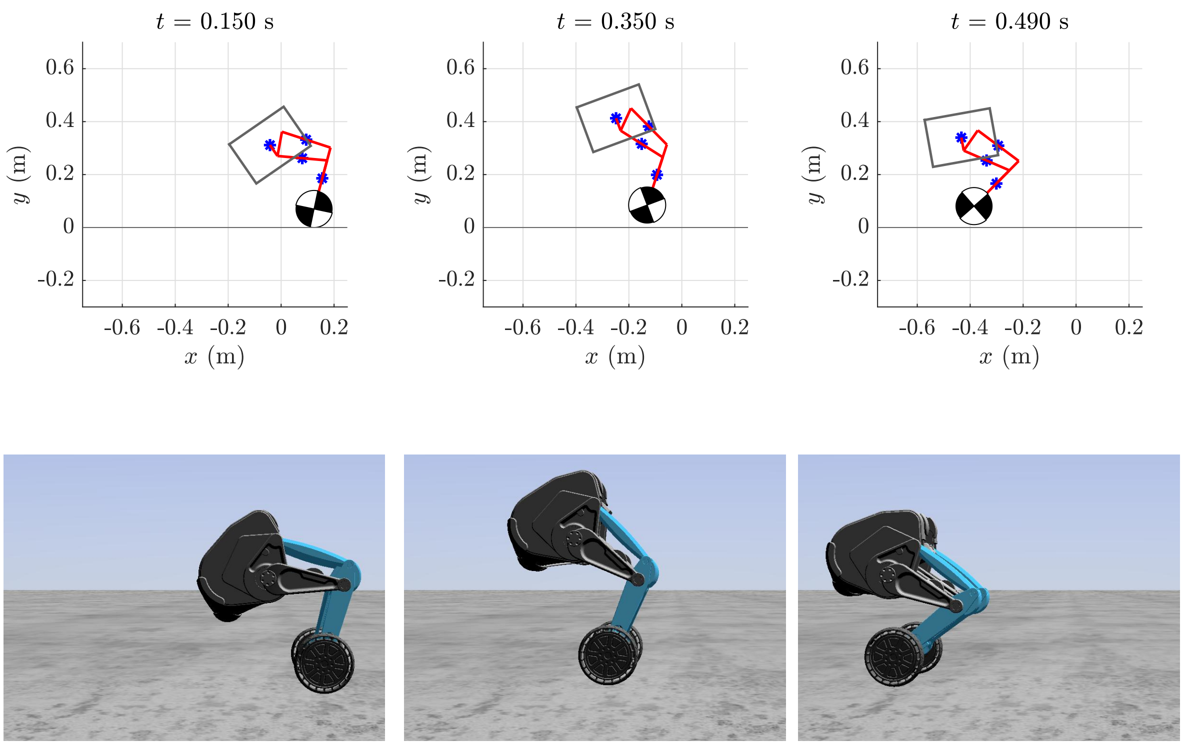


Figure 3.25: Forward jump — visual comparison of simplified model simulation (top) and 3D feedforward simulation (bottom).

4 | Real Robot Results

In this chapter, I present the results that I (and my colleagues) achieved with the real robot. We implemented the suggested control system from Chapter 3 without the UKF due to the limited time we had for experiments. The controllers (mainly the body motors PD regulators) were tuned to reflect the differences in parameters from the simulation model.

4.1 Experiments

In our experiments[1], we again tested the balancing and steering performance. The robot had no problem withstanding pushes from all directions. Snapshots of one push can be seen in Figure 4.1. As for the movement, the robot was capable of all kinds of different maneuvers, including fast stops or quick turns, following the references from a human-operated Xbox gamepad. Figure 4.2 shows that it even managed to go up a narrow ramp, turn around and go back down. We tried the experiments for different leg lengths and achieved decent results, but especially for more stretched poses, the performance was visibly worse and less robust.

The real robot performance exceeded my expectations in all aspects. The only thing that turned out to be of greater concern than we anticipated initially is the wheel slip. The wheels start slipping at torque values around 0.5 Nm, which significantly reduces the usable ± 2 Nm wheel motor torque range. Another (though expected) issue was the noise of motors velocity

[1]The recorded videos of the experiments are available at `https://youtu.be/B6OaJCYD5C8`.

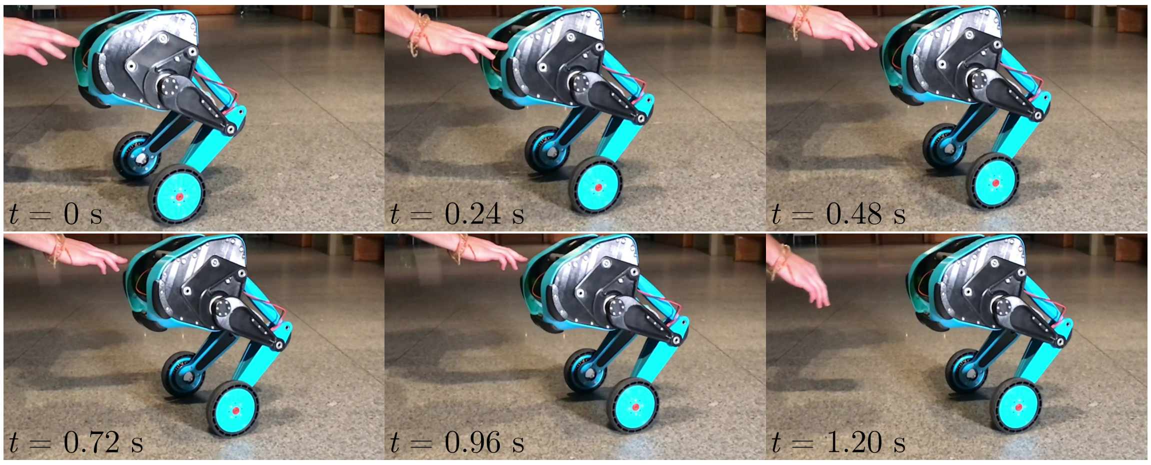


Figure 4.1: Real robot experiment — push.

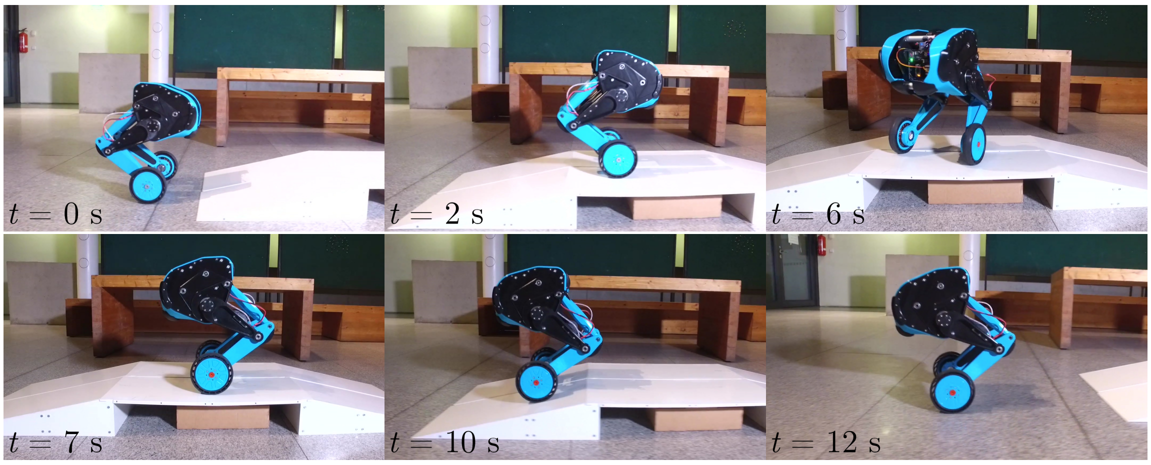


Figure 4.2: Real robot experiment — ramp turn.

and pitch rate measurements, preventing us from using faster, more aggressive control system tuning. This could be resolved by implementing the suggested UKF or by other filtering techniques.

4.2 Measurements

For the sake of completeness, I also present measured data from balancing and steering experiments.

Figure 4.3 shows measurements from a balancing experiment. The robot was initially pushed three times to its front, then hit three times from the side, and finally pushed four times to the back. The response is similar to the simulations. The main difference is that the side oscillations are damped quickly. This is probably caused by friction, which is, in reality, much higher than what was set in the 3D simulations. The steady state values of body motor torques u_4 were non-zero, which implies that the robot was not in equilibrium position. Furthermore, the peaks of u_4 are higher than in simulations, but that is caused by the different PD controller tuning[2].

Steering experiment data can be seen from Figure 4.4. The robot tracked the speed reference (denoted by the dark blue dotted line) with an approximately one-second delay, which corresponds to the simulations. The yaw rate reference tracking (denoted by the light yellow dotted line) was faster than in simulations with a delay of around 0.5 s. Leg weight load transfer occurred during cornering in agreement with the simulations, but no side oscillations are observed when the turning is stopped. Regarding the u_4 peak and steady-state values,

[2]The derivative gain was set lower than in simulations, thus the oscillation damping cannot be explained by the different tuning.

the same as for the previous balancing experiment holds.

The presented measurements only confirmed my claims about the performance from the previous section. The results are comparable to the simulations, except for side oscillations, which were quickly damped or did not occur at all. As I already mentioned, I explain this by differences in friction.

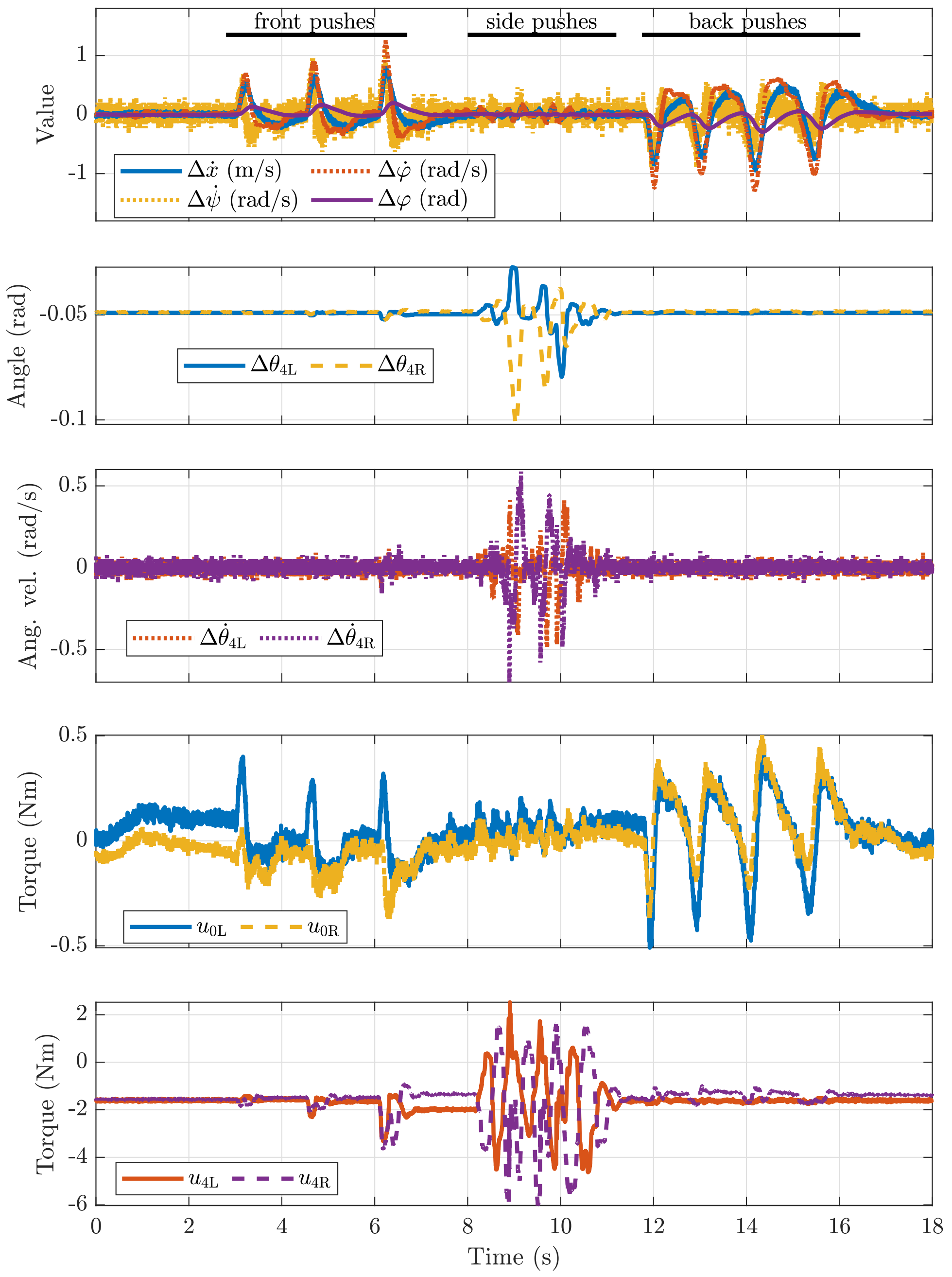


Figure 4.3: Real robot experiment data — balancing.

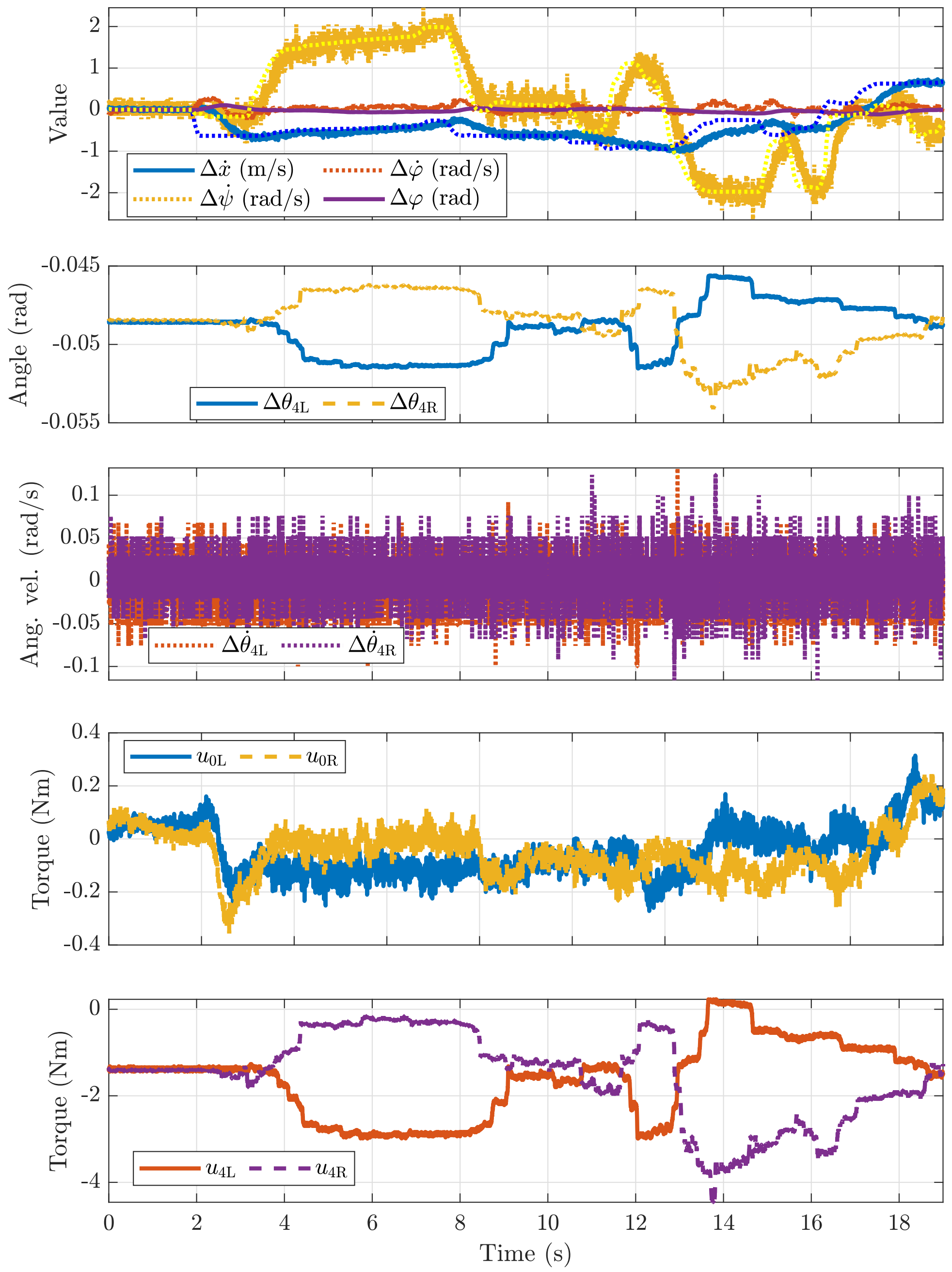


Figure 4.4: Real robot experiment data — steering.

5 | Conclusion

The goal of this thesis was to form basic simulation and control tools for our bipedal wheeled robot. In particular, to create a 3D simulation environment, design a control system, and implement the jump trajectory computation method based on the robot's planar model. I believe that I achieved the goal successfully.

In Chapter 2, I derived the planar model of the robot, which proved to be sufficiently accurate, and I used it for the jump trajectory computation and state observer design in Chapter 3. Moreover, I described the Simulink-Gazebo co-simulation that brings together the quick prototyping functionalities of Matlab/Simulink with Gazebo's 3D environment simulation tools. The 3D simulations served as the primary verification tool throughout the thesis.

Chapter 3 documents the control system. The planar model turned out to be insufficient for controller design due to couplings in yaw and roll dynamics. Thus, I switched to the two-wheel inverted pendulum model and used it to design an LQR with integral action for reference tracking. In combination with two body PD controllers, the LQR could stabilize and steer the robot with varying leg lengths. Furthermore, I designed an Unscented Kalman Filter, estimating the states of the planar model. The UKF worked well for balancing and straight motion, but the state estimates diverged from the true values during cornering due to unmodeled 3D dynamics in the planar model. In the last section of the chapter, I presented the optimization-based method for jump trajectory computation. The accuracy of the obtained trajectories was verified through simulation experiments, where the precomputed input values were used as a feedforward control sequence.

I described the experiments with the real robot in Chapter 4. The proposed controllers performed better than one could hope in terms of maneuverability, capable of quick turns while moving fast. The balancing performance was also great as the robot was able to withstand all kinds of pushes. The measurements only supported these claims and corresponded with the simulations, except for the unobserved side oscillations that were probably caused by different simulation friction settings.

5.1 Future Work

There is only a little room for improving the control system without the full 3D mathematical model of the robot. Thus, I see its derivation as the logical next step. Besides that, the method for jump trajectory computation needs to be upgraded. The manual tuning of the jump time is impractical and time-consuming. Lastly, the jump trajectories should be experimentally tested on the real robot.

References

[1] M. Bjelonic, C. D. Bellicoso, Y. de Viragh, D. Sako, F. D. Tresoldi, F. Jenelten, and M. Hutter, “Keep Rollin’—Whole-Body Motion Control and Planning for Wheeled Quadrupedal Robots,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2116–2123, 2019.

[2] M. Bjelonic, P. K. Sankar, C. D. Bellicoso, H. Vallery, and M. Hutter, “Rolling in the Deep – Hybrid Locomotion for Wheeled-Legged Robots Using Online Trajectory Optimization,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3626–3633, 2020.

[3] V. Klemm, A. Morra, C. Salzmann, F. Tschopp, K. Bodie, L. Gulich, N. Küng, D. Mannhart, C. Pfister, M. Vierneisel, F. Weber, R. Deuber, and R. Siegwart, “Ascento: A Two-Wheeled Jumping Robot,” in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 7515–7521, May 2019.

[4] V. Klemm, A. Morra, L. Gulich, D. Mannhart, D. Rohr, M. Kamel, Y. de Viragh, and R. Siegwart, “LQR-Assisted Whole-Body Control of a Wheeled Bipedal Robot With Kinematic Loops,” *IEEE Robotics and Automation Letters*, vol. 5, pp. 3745–3752, April 2020.

[5] C. Dario Bellicoso, C. Gehring, J. Hwangbo, P. Fankhauser, and M. Hutter, “Perception-less terrain adaptation through whole body control and hierarchical optimization,” in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pp. 558–564, 2016.

[6] K. Lynch and F. Park, *Modern Robotics: Mechanics, Planning, and Control.* Cambridge University Press, 2017.

[7] D. Cline, *Variational Principles in Classical Mechanics: 2nd Edition.* River Campus Libraries, 2018.

[8] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control.* Springer London, 2010.

[9] S. Kim and S. Kwon, “Dynamic modeling of a two-wheeled inverted pendulum balancing mobile robot,” *International Journal of Control, Automation and Systems*, vol. 13, no. 4, pp. 926–933, 2015.

[10] G. Franklin, J. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems.* Pearson Education, 2011.

[11] F. Lewis, D. Vrabie, and V. Syrmos, *Optimal Control.* Wiley, 2012.

[12] S. J. Julier and J. K. Uhlmann, "New extension of the Kalman filter to nonlinear systems," in *Signal processing, sensor fusion, and target recognition VI*, vol. 3068, pp. 182–193, International Society for Optics and Photonics, 1997.

[13] E. A. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pp. 153–158, 2000.

[14] Y. Wu, D. Hu, M. Wu, and X. Hu, "Unscented Kalman filtering for additive noise case: augmented vs. non-augmented," in *Proceedings of the 2005, American Control Conference, 2005.*, pp. 4051–4055, IEEE, 2005.

[15] D. Simon, "Kalman filtering with state constraints: a survey of linear and nonlinear algorithms," *IET Control Theory & Applications*, vol. 4, no. 8, pp. 1303–1318, 2010.

[16] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, 2014.

[17] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.