CZECH TECHNICAL UNIVERSITY IN PRAGUE

DIPLOMA THESIS

Variable-dynamic control of robotic leg with parallel mechanism

Author:
Bc. František KRÁČMAR

Supervisor:
Ing. Jaroslav BUŠEK, Ph.D.

A thesis submitted in fulfillment of the requirements for the degree of Diploma thesis

in the

Department of Instrumentation and Control Engineering

August 19, 2021

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kráčmar František** Personal ID number: **460008**

Faculty / Institute: **Faculty of Mechanical Engineering**

Department / Institute: **Department of Instrumentation and Control Engineering**

Study program: **Automation and Instrumentation Engineering**

Specialisation: **Automation and Industrial Informatics**

II. Master's thesis details

Master's thesis title in English:

Variable-dynamic control of robotic leg with parallel mechanism

Master's thesis title in Czech:

Robotická noha s proměnlivou dynamikou paralelního mechanismu

Guidelines:

1) Get acquainted with the Doggo robotic platform and similar projects.
2) Create a mathematical model of the five-bar linkage and identify its basic parameters.
3) Design the control of the mechanism so that it copies the prescribed dynamics.
4) Perform simulation and experimental verification of the proposed control.

Bibliography / sources:

[1] KAU, Nathan, Aaron SCHULTZ, Natalie FERRANTE a Patrick SLADE. Stanford Doggo: An Open-Source, Quasi-Direct-Drive Quadruped. In: 2019 International Conference on Robotics and Automation (ICRA) [online]. IEEE, 2019, 2019, s. 6309-6315 [cit. 2021-03-05]. ISBN 978-1-5386-6027-0. Dostupné z: doi:10.1109/ICRA.2019.8794436
[2] BLACKMAN, D. et al. (2016). "Gait Development On A Direct Drive, Quadrupedal Robot".
[3] SPONG, Mark W., Seth HUTCHINSON a M. VIDYASAGAR. Robot modeling and control. Hoboken: John Wiley, 2006. ISBN 0471649902.

Name and workplace of master's thesis supervisor:

Ing. Jaroslav Bušek, Ph.D., U12110.3

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

Date of master's thesis assignment: **30.04.2021** Deadline for master's thesis submission: **24.08.2021**

Assignment valid until: _____

Ing. Jaroslav Bušek, Ph.D. Supervisor's signature	Head of department's signature	prof. Ing. Michael Valášek, DrSc. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt	Student's signature

Declaration of Authorship

I, Bc. František KRÁČMAR, declare that this thesis titled, "Variable-dynamic control of robotic leg with parallel mechanism" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"I'm smart enough to know that I'm dumb."

Richard P. Feynman

CZECH TECHNICAL UNIVERSITY IN PRAGUE

Abstract

Faculty of Mechanical Engineering

Department of Instrumentation and Control Engineering

Diploma thesis

Variable-dynamic control of robotic leg with parallel mechanism

by Bc. František KRÁČMAR

This thesis deals with the implementation of a robotic leg with a parallel mechanism. Improvements to the mechanical design, which is based on a publicly available model from Stanford University, are described. The thesis also discusses the implementation of a mathematical model of the five-arm mechanism and the design of such a control so that the mechanism has the prescribed dynamics. Another method of controlling the dynamics is based on cascading PID controllers. The proposed control strategies are experimentally verified on a physical leg.

Keywords: robotic leg, five-bar linkage, variable dynamics, full state feedback, ODrive
Klíčová slova: robotická noha, pětiramenný mechanismus, proměnlivá dynamika, stavová zpětná vazba, ODrive

Acknowledgements

I would like to thank my supervisor, Jaroslav Bušek, for inviting me as his right hand, getting me started down the path of mobile robotics; for the freedom and resources to explore this area; and everything else.

Then I want to thank the CTU which gave more than 6 years of great experiences.

Last but not least I my thanks go to my mother, my sister and to my girlfriend Kateřina for their relentless support.

This work was supported by a grant from the CTU Student Grant Competition nr. SGS20/159/OHK2/3T/12.

Contents

List of Figures

List of Tables

Listings

Glossary

BLDC Brushless DC electric. 8

CAD Computer aided design. 1

CAN Controller area network. 13

CTU Czech Technical University in Prague. 1

DARPA Defense Advanced Research Projects Agency. 2

ETH Eidgenössische Technische Hochschule. 4

LiPo Lithium polymer. 12

MIT Massachusetts Institute of Technology. 4

PCB Printed circuit board. 13

PLA Polyactic acis. 8

PM brushless Permanent magnet brushless. 31

PWM Pulse width modulation. 14

SLS Selective laser sintering. 8

SPI Serial peripheral interface. 14

UART Universal asynchronous receiver-transmitter. 13

USB Universal serial bus. 13

Physical Constants

Gravitational acceleration $g = 9.81\,\mathrm{m/s^2}$

List of Symbols

a	acceleration	m/s^2
A	state matrix	
A_t	measured amplitude envelope value at time t	
b	damper coefficient	N/m^2
B	input matrix	
B_1	friction coefficient	N·m/(rad/s)
C	output matrix	
D	feedforward matrix	
F	force	N
H_c	gain of current transducer	
I	moment of inertia	kgm^2
J	moment of inertia	kgm^2
k	spring coefficient	N/m
K	feedback gain	
K_a	induced emf constant	V/m/s
l	length	m
L	observer feedback gain	
L_a	(L-M) in PM brushless DC machine	H
m	mass	kg
q	independent variable	
R_a	twice per phase resistance	Ω
t	time	s
T_1	electromagnetic force	N
u	manipulated variables	
x	state variables	
y	output variables	
ξ	damping ratio	[-]
τ	torque	Nm
ω	angular speed	rad/s

Chapter 1

Introduction

My thesis deals with the development of a robotic leg and it's control algorithms.

Whole thesis is divided into several chapters. In the chapter 2, I described the evolution of mobile robotics, introduced similar projects and made an overview of approaches to dynamics management on mobile robots.

Following chapters, chapter 3 and chapter 4 are intended to be an introduction to Doggo platform and describe my own leg design. The first task was the mechanical design of the leg itself. It was based on a publicly available CAD model from Stanford. However, as you can read in the chapter 3, their design had several major flaws that I fixed, also some parts were simplified.

The chapters that follow focus on the design of leg control to copy the prescribed dynamics. In the chapter 5, I took a more theoretical approach to the task. I constructed a mathematical model of the dynamics of the five-bar mechanism and identified its basic parameters. Subsequently, I designed a full state feedback controller and performed its experimental validation. In the chapter 6, I used an experimental approach. I have verified that it is possible to change the dynamics of a BLDC motor by cascading PID controllers. Based on the experimental data for different controllers parameters, I developed a method that determines new parameters for prescribed dynamic behavior.

Robotics is so complex domain, that I could not handle all task accompanied with robotic leg and I did not try to do so. Robots mentioned in chapter 2 are group projects with time span across many years and secured funding. My thesis was work done by a single author during my master's degree on CTU. This led to some simplifications, assumptions and take overs. Biggest inspiration to me was Doggo project done by Stanford university. They have born in mind not only financial side of whole project, which can be crucial for such capable robot, but also have made their mechanical, eletronical and software solutions fully available.

My main goal was to built a fully functional version, which other students can use to push it's boundaries further.

Chapter 2

Mobile robotics overview

Mobile robotics plays and is going to play a big role in our lives. Thanks to their evolution, robots can perform many tasks that help us. I would like to dedicate first part of my thesis to present mobile robotics as a domain and it's components. Furthermore, this chapter contains an overview of approaches to dynamics management in similar projects to my thesis.

2.1 Introduction to mobile robotics "How we got here"

More than 1 million years ago, early humans began to make stone tools. They were pretty smart, because they knew that with tools they could do more with less energy expenditure and in less time. This feature evolved with human being and we gradually tamed fire, metals and power of steam. This led us to building more sophisticated tools than ever and those "machines" started doing tasks instead of us, from moving heavy stuff to doing billions of operations during the blink of an eye.

In recent decades, scientists and engineers have begun working on machines that combine mechanical capabilities with computer control. This starts with having more motivators. Let me mention, for example, the demographic development of Japan, when it is clear that the population is aging overall and it was clear to the officials that it will be necessary to develop devices that will take care of the old. Another motivator for development is the demand for machines that can withstand human activities in unsuitable conditions - such as in the rubble after natural disasters.

Individual attempts did not have a way to measure which one was objectively better. And since robotics consist of many individual parts, a good way to do this is to compare robots against high level goals. Many international competitions have taken places for this purpose. Probably the most prestigious have been organised by DARPA. To name a few, DARPA Grand Challenge gave a huge playground to American Innovators in early years 2004 and 2005. Their goal was to build first fully autonomous vehicle which could complete an off-road trail in a limited time. DARPA Robotics Challenge then, between years 2012 and 2015, presented obstacle course for humanoid robots.

Some competitions have been organised to bring this type of atmosphere and enthusiasm among students. For example The ARLISS Project, which takes place in Nevada's desert and simulates orbital mission to Mars, is students competition for students from around the worlds. Similar competition takes place on the other side of the Pacific Ocean. The China Adolescent Robotics Competition, which started in

2001, brings China's youth to the robotics via task to build their own robots and solve fairly complex tasks, such as placing plastic hedgehogs on a thin piece of metal.

2.2 Scopes of mobile robotics

When we take a look at mobile robotics these days, we see many domains that have to be mastered (Institute, 2018). Enumeration starts with mechanical design and goes to high level control, etc. You can see a mind map illustrating domains in fig. 2.1

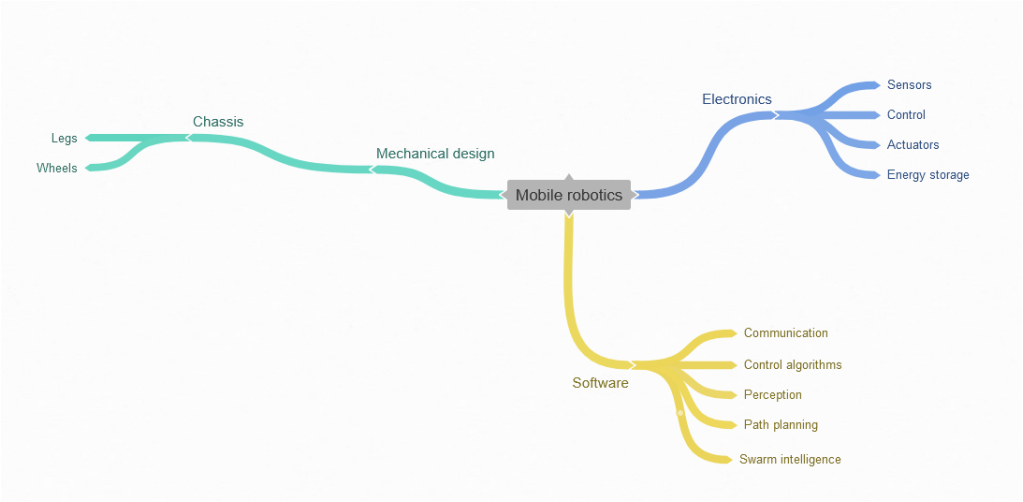


FIGURE 2.1: Scopes of mobile robotics

A robot is, in its original sense, a mechanical machine. And as such, it has a mechanical chassis that needs to be designed and manufactured. Knowledge from the design of structures, production technology, materials (especially composites) is used here.

Along with the mechanical construction, the robot also contains a large amount of electronics. It serves several purposes. First, using a computer or microcontroller, it controls the movement of the robot itself. Either the robot can be controlled remotely by the operator and the computer provides control of the actuators. Or there is a control algorithm that is implemented directly in the computer, and the robot does not need control instructions from the outside - in this case we are talking about an autonomous robot. Another purpose of using electronics are actuators. These are mainly electric motors - rotary or linear - in special cases piezoactuators. All these electronics must have their own power supply. If the robot is not clearly mobile, it can be supplied with energy by a power cable. When this is not possible - typically the robot moves in free space - it must carry the energy source with it. Energy is stored in batteries, which are attached to the chassis of the robot.

Another big part of robotics is software. Almost every machine today already communicates with its environment in some way. Communication is also a cornerstone for the development of Industry 4.0, where the transfer of information between things is key. The microcontrollers then run the programs that control the entire robot. Here, SW practices such as state machine, full state feedback, Kalman filter or path planning are applied. An interesting SW field is swarm intelligence. In many cases, the robot will not perform its actions alone, but will cooperate with

other robots and machines. A current topic is, for example, the communication of autonomous driving in convoy (Hobert et al., 2015). We call such communication "V2V" or "Vehicle-to-Vehicle", which falls under "V2X" ("Vehicle-to-Everything") communication.

The robot in autonomous mode is directly dependent on information about its surroundings. Operator-controlled robots are then usually equipped with cameras to control them remotely. A wide range of sensors is used here, from position, force, pressure sensors to the cameras or lidars, for example.

2.3 An overview of current mobile legged robots

My diploma thesis is based on the Doggo robot, which can be seen in the figure 2.2. This is a student project that was completed in 2019. Its advantage is the relatively low price ($ 3,000). For example, a Chinese copy of MIT's mini cheetah robot, which will be mentioned below, costs $ 16,000. It is also very beneficial that the whole project is open to the public. You can take over and customize the design of the chassis, electronics and software. The robot itself is composed of a large number of off the shelf parts, only specific parts are made to order.

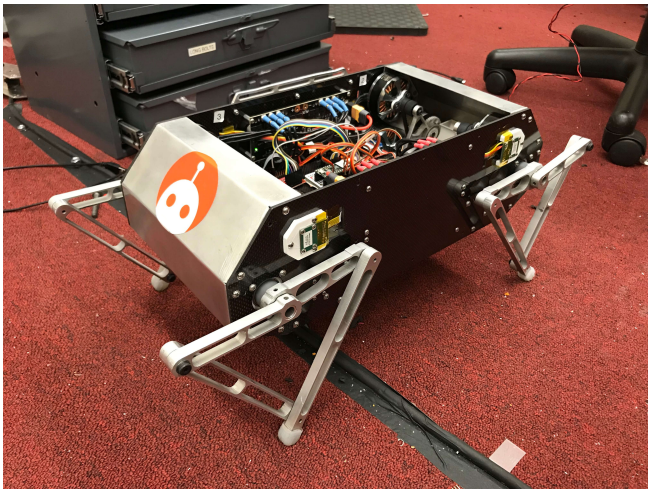


FIGURE 2.2: Stanford's Doggo (Daily, 2021)

Another student project is the mini cheetah introduced at MIT in 2019 under lead developer Benjamin Katz. The advantage of this robot is that it is robust and doesn't break easily, and if it does it can be fixed easily because it is built with modular parts. What is not usual with student robots is that mini cheetah motors and their controllers have been specially developed for this project. The robot can be seen in the figure 2.3.

ANYmal from ETH Zurich in Switzerland can be considered as a representative of European mobile robots. It is definitely worth paying attention to for the fact that it does not have to move only by walking, but has wheels at the end of its legs that can make it drive or combine these movements in "Hybrid Locomotion" (Marko Bjelonic and Hutter, 2020). This platform is already mature and stable enough to be offered for industrial applications.

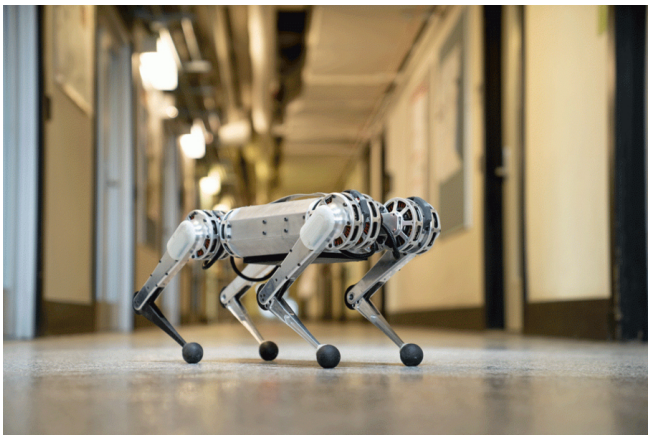


FIGURE 2.3: MIT's mini cheetah (News, 2021)

FIGURE 2.4: ETH's ANYmal (Zurich, 2021)

Probably the most famous robot of today is Spot (in figure 2.5) from Boston dynamics. This American company has many years of experience in developing cutting-edge robotic technologies. They have moved from clunky "walking cages" to humanoid robots that can flip and dance (Dynamics, 2020). Spot can be fitted with a robotic arm that allows it to further interact with its environment - for example, it can open doors or carry objects. Spot is already in commercial use. Companies that use it include NASA Jet Propulsion Laboratory, STRABAG, BP or Czech Alza.

2.4 Approaches to dynamics management in similar projects

In this section I will give a brief overview of leg movement control approaches for projects similar to my thesis.

I'll start, of course, with my "predecessors" - the creators of the Doggo robot (Kau et al., 2019). Their approach to controlling the leg dynamics was similar to mine in chapter 6. The dynamic behavior was provided by PID controllers implemented in ODrive. They used the proportional term of the controller to add virtual stiffness and the derivative term to add virtual damping.

FIGURE 2.5: Boston Dynamic's Spot (Dynamics, 2021)

The very simplest way to control leg movement was used by students at FAMU FSU College of Engineering (Blackman et al., 2016). For steering, they are concerned only with the kinematics of the motion. However, they augment this with end effector forces, which they use as feedback to perceive the surrounding environment.

Previously, there has also been an attempt (Hutter et al., 2013) to control the dynamic behaviour of the foot mechanically. This principle was referred to as "Series elastic actuators" and consisted of a unidirectional spring and damper as part of the leg, as can be seen in the figure 2.6.

They also describe here the widely used mathematical model "Spring-Loaded-Inverted-Pendulum", which simplifies the robot's mass point supported by a compliant leg. This model is used to control the dynamics of the legs.

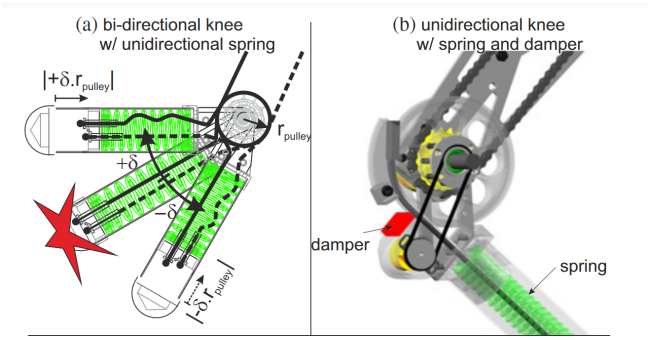


FIGURE 2.6: Series elastic actuators leg (Hutter et al., 2013, edited)

As an alternative to the "Spring-Loaded-Inverted-Pendulum" model, where the stiffness of the leg is primarily controlled, an approach is described here (Park, Chuah, and Kim, 2014) where the motion of the leg during walking is defined by the time of contact with the ground and the time the leg is in the air. This ensures the character of the movement - for example, gait or trotting. A sample time course of such a movement can be seen in the figure 2.7.

In this context it is interesting to compare the normal pad contact time of a human, which is 150 ms, and that of a snow leopard, which is significantly shorter at 45 ms.(Institute, 2018)

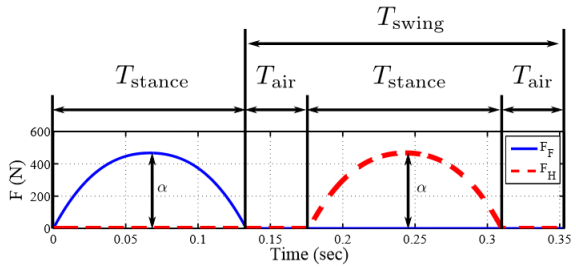


FIGURE 2.7: Bounding Control with Variable Duty Cycle (Park, Chuah, and Kim, 2014)

Chapter 3

Design of robotic leg mechanics

I devoted a significant amount of work on the mechanical design to the possibility of manufacturing the leg. This means checking all dimensions to match European ones and modifying some. It is necessary to mention that the CAD model of Stanford was not complete, some parts were completely missing. For example, I had to invent a radial bearings for pulleys. Some parts were not assembled at all. I solved such things in my model as well. The Doggo's documentation (Kau, 2020) contains a bill of materials and standardized components. But it no longer contains assembly instructions and some items have only a short description of what function they are used for. It took me a long time to assign them correctly. I have created this detailed bill of materials to ensure smooth production and to be able to buy all components in the Czech Republic. However, Doggo project is cost effective thanks to maximal use of off the shelf components. While other robots, such as the previously mentioned Cheetah for example, already have custom-made motors. (Katz, 2018)

The mechanism as a whole can be seen in the figure 3.1. The leg linkage (number 1 in the figure) consists of 4 segments that form a closed loop. Two segments are actuated by an "inner drive shaft" (number 2) and an "outer drive shaft" (number 3). These shafts are coaxial. Torque is supplied to the shafts from two BLDC motors (number 4) by belt transmissions (number 5). The gear ratio has been chosen to be 1:3 (16:48 teeth pulleys (number 6 and 7)). This is done in order to gain torque with smaller motor mass. At the same time, the ratio is not too large so that the dynamics of this belt transmission is as small as possible and proprioception can be done in the motors and not on the leg linkage. With a too high ratio, for example, the transfer of moments from the leg linkage to the motors would not be possible and the mechanical design would have to be made more complex. The "side plate" (number 8 in the figure) is the side of the robot and serves to mount the whole mechanism. The brace (number 9) is used to provide radial clearance between the motors and dive shafts.

Adjusting screws are used to transfer the torque from one of the 3D printed pulleys to the shaft and its axial securing, which are mounted in the 48T pulley by means of threaded inserts. However, the dimensions of the pulley and the insert do not fit in the available CAD model Doggo, so I remodeled the pulley and adjusted the necessary dimensions. Due to the high mechanical stress of the pulleys, I had them made by a 3D printing method called "selective laser sintering" (SLS) from powdered nylon, which provides better mechanical properties than same pulley made with Fused deposition modeling from standard PLA material. Final design of a remodeled pulley can be seen in fig. 3.2, where edited parts are highlighted by red circles.

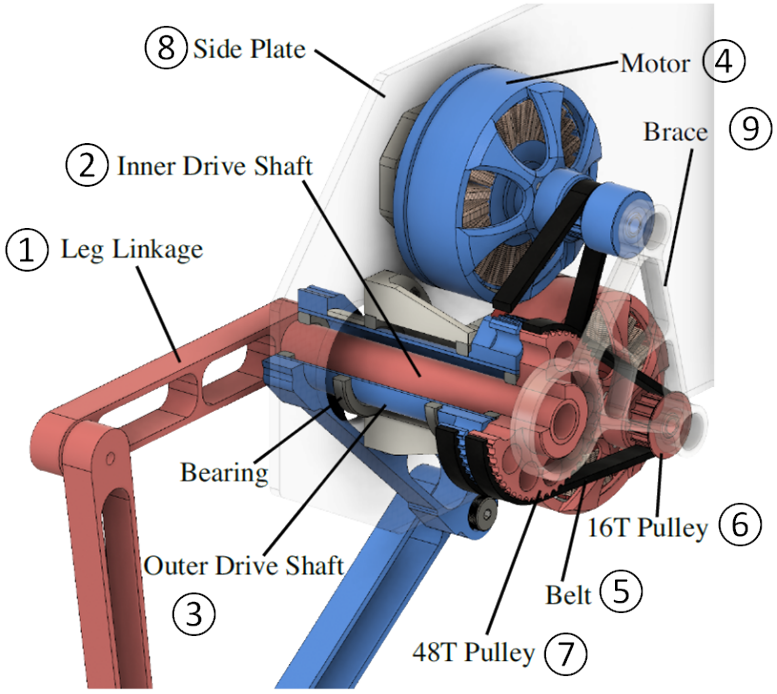


FIGURE 3.1: Coaxial shafts (Kau et al., 2019, edited)

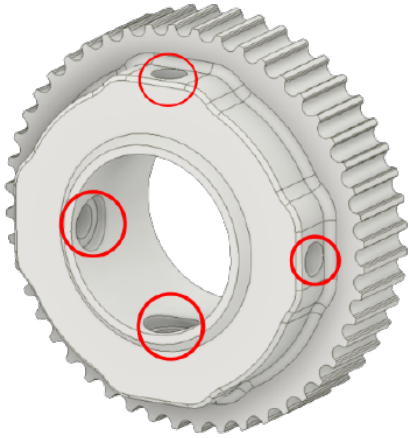
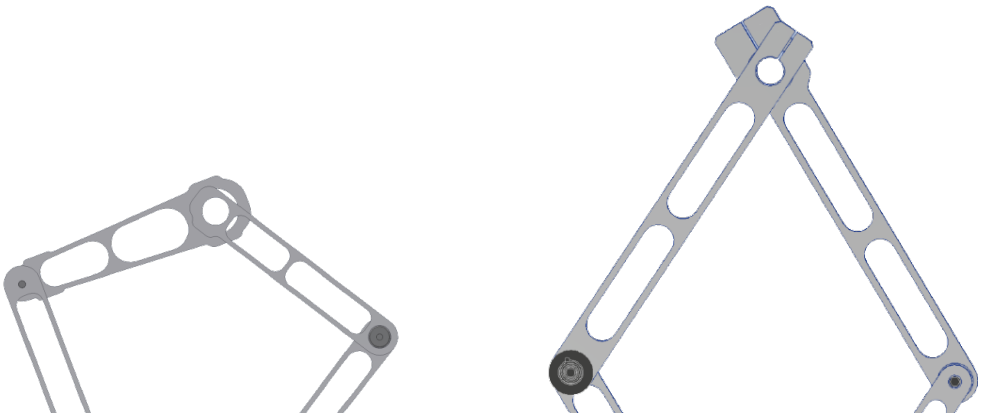


FIGURE 3.2: Remodeled pulley

Probably the biggest change in mechanical design has been performed on clamping joint between leg and driven axis. Solution invented by Stanford (see figure 3.3a) uses only adjusting screw. Which seems to work, according to their public materials (Stanford, 2019). However, since my robot should last as long as possible, I preferred a more robust clamping joint (see figure 3.3b).

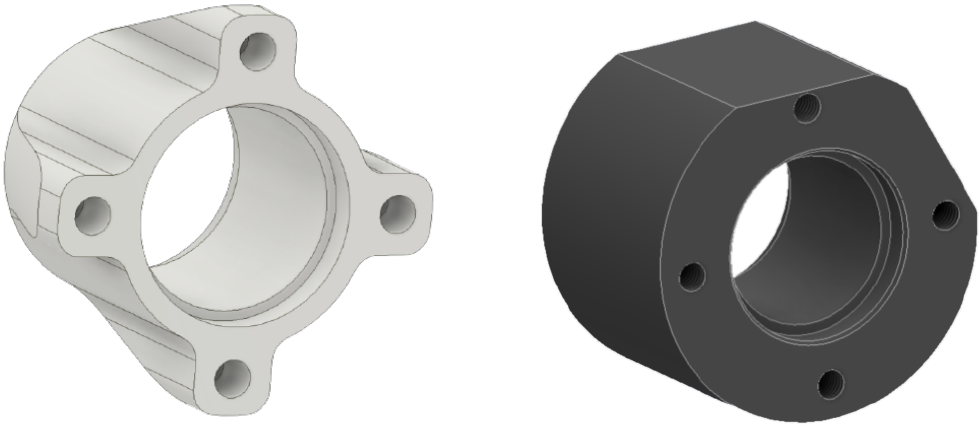
Last but not least, I designed a new housing for attaching the entire leg assembly to the frame. The original design (fig. 3.4a) was effective, but very demanding for production (use of a milling machine, etc.). Therefore, I took over only the main



(A) Stanford's version (B) My version

FIGURE 3.3: Old and new versions of the clamping joint

dimensions (e.g. to be able to insert the designed bearings) and proposed a structurally significantly simpler solution. The resulting housing can be produced almost completely on a lathe and can be seen in the figure 3.4b.



(A) Stanford's version (B) My version

FIGURE 3.4: Stanford's and my version of the housing

After evaluating the production and during the assembly, I came to a few remarks that should be taken into account when creating another version of the leg.

Firstly, the clearance of the bolts transmitting the torque to the outer shaft is determined by the (in)accuracy of the manual assembly. Two things go against each other here: manual assembly and minimal clearance. In certain cases, the resulting clearance will be non-zero, thus creating hysteresis, which will be reflected in the dynamics.

SLS pulley production is not the cheapest and the mechanical properties can only be verified experimentally, so it would be appropriate to consider production from a pulley stock bar for other versions. This bar already has the necessary dimensions for the selected belt and all that remains is to finish the required mechanical connection. Example of such rod can be seen in figure 3.5.

FIGURE 3.5: Pulley stock bar (Tyma CZ, 2020)

Chapter 4

Design of robotic leg electronics

In this section, i will briefly touch on electronics used in Doggo's legs. I decided to use the same electronics as they did because it has been tested to work as a whole, a mechanical bearing design has been prepared for it and it has good dynamic capabilities. Electronics consists of three parts only - the main control board, magnetic encoders, brushless DC motors and LiPo battery pack. The electronics diagram can be seen in the figure 4.1. One of the following subchapters is devoted to each of the components.

The electronics form a closed control loop. ODrive represents a regulator that controls the action variable - the voltage on the motor - based on the setpoint. The position of the motor shaft is then measured using a magnetic encoder and sent to the ODrive as a feedback.

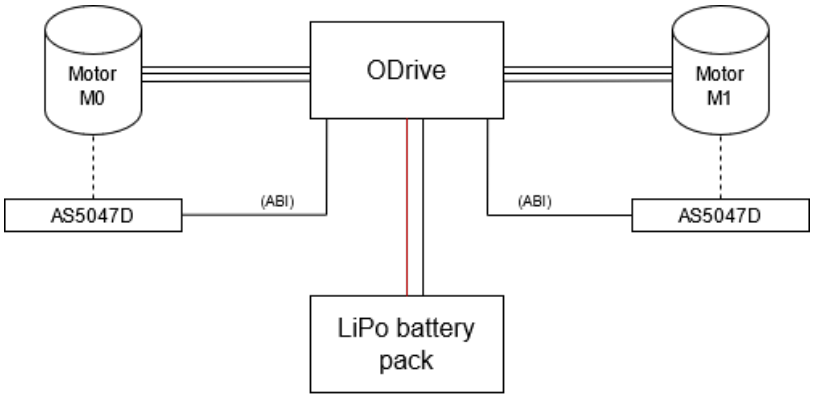


FIGURE 4.1: Diagram of the overall connection of electronics

4.1 Control board - ODrive

The main electronic component of the Doggo robotic leg is the ODrive motor controller. It is controller board for high power (120 A peak current per motor) brushless motors intended for hobby applications. ODrive was open source up to version v3.5 (both hardware and software), when current version v3.6 is closed-source with respect to board files and schematics.

The board can be seen in the figure 4.2. It is built on a arm-based microcontroller and controls up to two brushless motors at the same time.

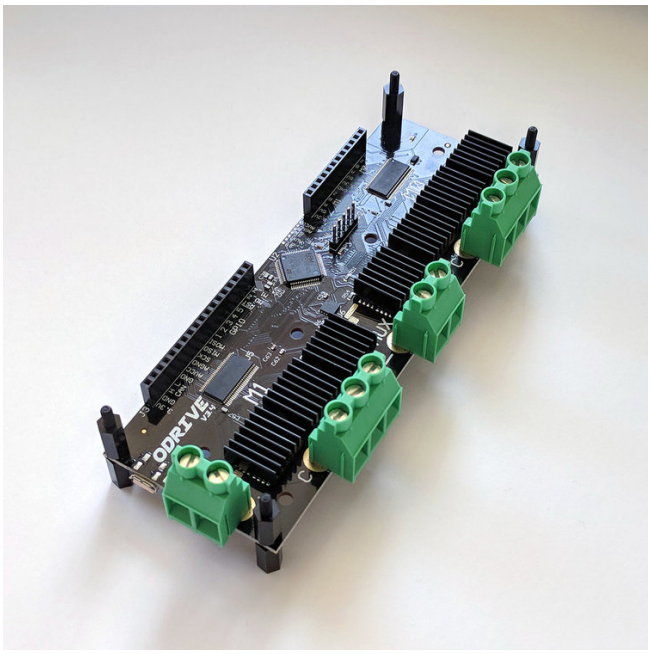


FIGURE 4.2: ODrive board (Robotics, 2021b)

Its advantages over competing projects include: accessible complete documentation, an extensive user community that helps each other, and increasing new features. On the other hand, for example, the moteus platform from mjbots is more compact. Unlike ODrive moteus's PCB, it already contains encoders and only the motor, power supply and CAN bus are connected to the board.

Another advantage of ODrive board is the wide range of developed functions. The motors have position, velocity, and current control modes. The board automatically identifies the motor parameters. As for communication, ODrive can be controlled via real-time USB communication from Python, UART communication using the developed Arduino library or CAN.

TABLE 4.1: Specifications of ODrive v3.6 56 V

Attribute	Value
Input voltage	12 V to 56 V
Interfaces & Protocols	USB, UART, CAN, PWM
Max continuous current with heatsink in still air	40 A per channel
Dimensions	135.5 x 50 mm

4.2 Absolute position sensors - AS5047D

Feedback of motor shafts to ODrive is secured via absolute position sensors. AS5047D are used in this particular application. They are 14-bit magnetic rotary position sensors suitable for high speed measurements. I used the sensors in the so-called "Evaluation Board" version, where the sensor is supplied on a printed circuit board with

lead-out pins for powering the wires. This solution (which can be seen in the figure 4.3) is advantageous for possible replacement of the sensor, but above all the plate facilitates the mounting of the sensor, because it has holes for screws. One limitation had to be taken into account when choosing an encoder, namely a supported interface. Although ODrive has SPI and PWM hardware interfaces, at the time of the selection these interfaces were not supported in the firmware, so it was more appropriate to choose encoders with the ABI interface than to develop your own firmware version. At the time of writing this text (July 2021), there is already a new firmware version that supports SPI and PWM. Main specifications can be found in table 4.2.

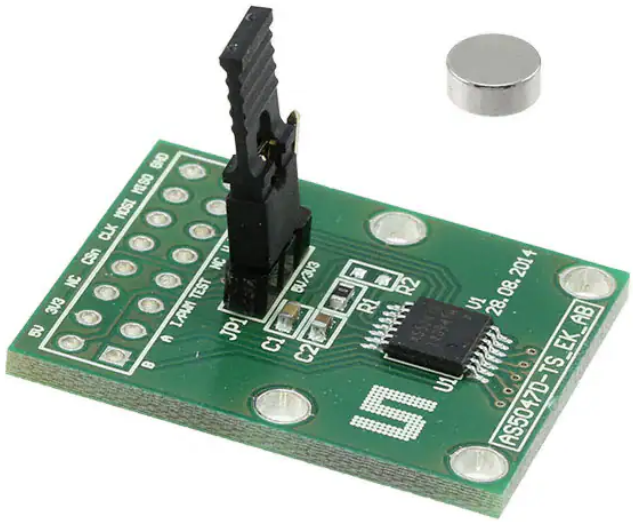


FIGURE 4.3: Evaluation board AS5047D-TS_EK_AB (Digi-Key, 2021)

TABLE 4.2: Specifications of AS5047D

Attribute	Value
Position resolution	14-bit
Interface	SPI, ABI, UVW, PWM
Operating Supply Voltage	3.3 V, 5 V
Weight	22 g

4.3 Brushless DC motors - T-motor 5212

Brushless DC motors T-motor 5212 are used as main actuators in robotic leg. Since two links are powered, two motors are needed for one leg. Brushless motors are usually used for high speed applications - copters, etc. However, they can also be used here at low speeds thanks to the advanced control in ODrive.

One motor is shown in the figure 4.4. It can be seen that it has a relatively small height, which is an advantage especially for the design of the chassis - it has less demands on installation space. As for the performance of this motor, from the dynamic presentations of the Doggo (Kau, 2021), it is sufficient for jumps with a robot up to a height of 1 m. Maximal continuous power of T-motor is 840 W as can be seen in

table 4.3 with other specifications. It's maximal continuous current is 35 A, which is less than ODrive can handle and battery pack can deliver.

FIGURE 4.4: T-Motor MN5212 (3DXR, 2021)

TABLE 4.3: Specifications of T-motor 5212 KV340

Attribute	Value
KV	340 rpm/V
Motor diameter	59 mm
Motor length	33.5 mm
Weight	205 g
Idle current	1.1 A
Max continuous current	35 A
Max continuous power	840 W
Internal resistance	69 Ω

4.4 LiPo battery pack

The main power source for leg is LiPo battery pack. A classic battery used for RC models, such as a quadcopter, was used. However, sufficient capacity was an important requirement for longer-term testing. If a low-capacity battery pack were used, it would discharge rapidly, reducing its voltage, and this could affect the overall dynamic capabilities of the motors. The selected LiPo akupack consists of 6 cells, as required by the motor. The battery minimal working voltage is 22.2 V and 25.2 V when fully charged, which is sufficient for ODrive. The maximum load of the battery pack can be up to 159 A, which is enough for the demands of the motors. Main attributes of used akupack can be found in table 4.4. The specific battery that has been used can be seen in the figure 4.5.

FIGURE 4.5: Used LiPo battery pack

TABLE 4.4: Specifications of LiPo battery pack

Attribute	Value
Voltage	22.2 V
Cells	6
Capacity	5300 mAh
Max load	159 A (30 C)

Chapter 5

Proposed control based on mathematical model of the five-bar mechanism

In order to control the dynamic mechanism with applied knowledge from control theory, I had to build a mathematical model of this system.

First of all, I would like to describe how the mechanism looks like in terms of dynamics. Its idealized scheme can be seen in the figure 5.1, where I have replaced each arm with a perfectly rigid beam with all the mass concentrated in the center of gravity. The linkage consists of 4 arms that have lengths l_1 to l_4, masses m_1 to m_4, moments of inertia I_1 to I_4, and distances of the center of gravity from the joints l_{c1} to l_{c4}. I have placed the x-y coordinate system in the mechanism to frame connection - the joint of arms 1 and 2. These arms are driven completely independently by external moments τ_1 and τ_2, while arms 3 and 4 move dependently of these moments. In my text, I refer to the joint between arms 3 and 4 as the end-effector. Linkage is in the general position in the picture. Typically, the angles q_1 and q_2 will be around 270°.

5.1 Mathematical model of the specific five-bar mechanism - paralelogram

Configuration in figure 5.1 with two driven segments is called five-bar linkage and is one of the commonly used parallel mechanisms, because it provides greater rigidity of the mechanism and also lower weight of the moving parts due to the possibility of placing the drives in the base frame. In this section I formulate a mathematical model describing this mechanism.

I started from the mathematical model described by Spong (Spong, Hutchinson, and Vidyasagar, 2005) and added a damping matrix. I further linearized this model and designed its control in next sections.

The derivation of this model will first describe the positions of the centers of gravity of each segment and their derivations. Subsequently, after defining the Jacobians can be fitted to the inertia matrix. These steps are not listed here; the reader can consult the literature for them. However, I describe the following steps.

The inertia matrix is given by:

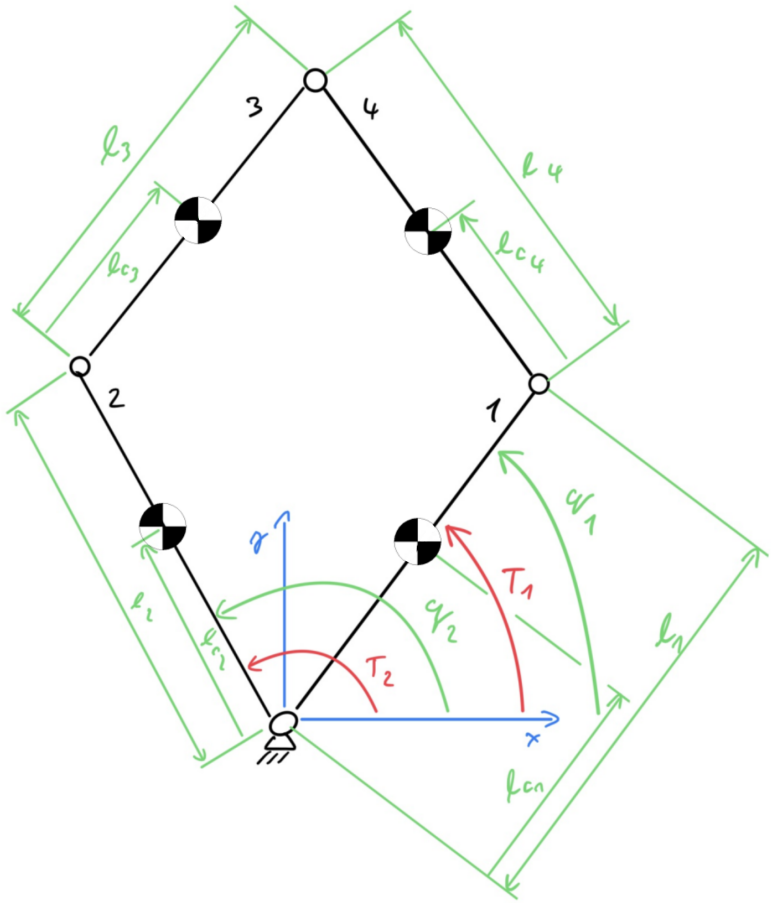


FIGURE 5.1: Linkage scheme

$$\mathbf{D}(q) = \sum_{i=1}^{4} m_i J_{vc}^T J_{vc} + \begin{bmatrix} I_1 + I_3 & 0 \\ 0 & I_2 + I_4 \end{bmatrix}. \tag{5.1}$$

Where the individual elements have following forms:

$$\begin{aligned} d_{11}(q) &= m_1 l_{c1}^2 + m_3 l_{c3}^2 + m_4 l_1^2 + I_1 + I_3 \\ d_{12}(q) &= d_{21}(q) = (m_3 l_2 l_{c3} - m_4 l_1 l_{c4}) \cos(q_2 - q_1) \\ d_{22}(q) &= m_2 l_{c2}^2 + m_3 l_2^2 + m_4 l_{c4}^2 + I_2 + I_4. \end{aligned} \tag{5.2}$$

The potential energy of the mechanism is given by

$$\mathbf{P} = g \sum_{i=1}^{4} y_{si} \tag{5.3}$$

and it's partial derivatives are

$$\begin{aligned} \frac{\partial \mathbf{P}}{\partial q_1} &= g_1 = g\cos(q_1)\left(m_1 l_{c1} + m_3 l_{c3} + m_4 l_1\right) \\ \frac{\partial \mathbf{P}}{\partial q_2} &= g_2 = g\cos(q_2)\left(m_2 l_{c2} + m_3 l_2 - m_4 l_{c4}\right). \end{aligned} \tag{5.4}$$

Which are then combined into single vector $\mathbf{g}$ as

$$\mathbf{g} = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}. \tag{5.5}$$

Now, lets define τ as vector of external torques.

$$\tau = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \tag{5.6}$$

Equations 5.1, 5.5 and 5.6 are then written into Euler-Lagrange equation in it's common form

$$\mathbf{D}(q)\ddot{q} + \mathbf{C}(q,\dot{q})\dot{q} + \mathbf{g}(q) = \tau. \tag{5.7}$$

Where matrix $\mathbf{C}$ is damping matrix.

Spong states the condition that segments 1 and 2 are not dynamically affected. However, in my case it is not fulfilled due to the asymmetrical weight distribution and it is not possible to conveniently divide the control into individually driven segments.

Damping is not considered for this Spong's mdoel, however I put the friction caused by the bearings into damping matrix $\mathbf{C}$. Individual components of bearing friction and their action are described, for example, by Zughaibi (Al-Zughaibi, 2020). How bearing friction changes with angular velocity can be seen in fig. 5.2. The friction of the belt drive, which connects the pulleys on the individual motors and the coaxial shafts, is not considered. BLDC motors operate in such torque and speed ranges that both belt friction and weight can be neglected.

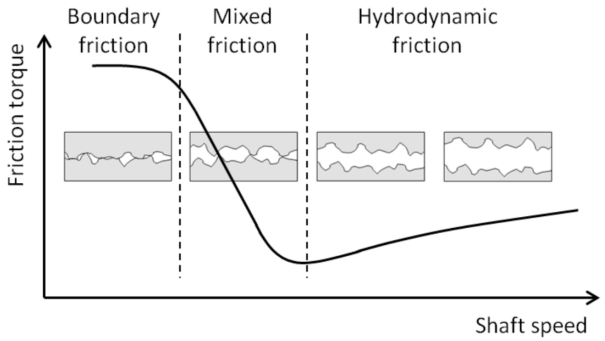


FIGURE 5.2: Dependence of bearing friction on angular velocity (Sander et al., 2016)

5.2 Determination of physical parameters of the five-bar mechanism

Model described before is in general form - physical parameters are represented by variables. In order to tune proposed control strategy, I had to replace these parameters with real values of this leg.

The individual leg segments are described in the model by their weights, moments of inertia, lengths and distances of the centers of gravity. To determine these parameters, I used a CAD program in which I performed the proposed mechanical improvements (Autodesk Inventor 2020). I determined the actual density from the weights of the manufactured segments. For example the measured weight of segment 1 is 55 g and its volume given by Inventor is 20.1208 cm^3. This gives us the desired density 2.7335 g/cm^3 after dividing weight by volume. These parts are made out of aluminium which is already included in Inventor's material library with density 2.7 g/cm^3. I chose density of aluminium from material library for further measurements. Housing together with bearings and pins are made out of steel and their density was therefore set to 7.85 g/cm^3.

Furthermore, it was necessary to take into account the weights of the pins, which were not negligible but not considered in the mathematical model. I decided to add half the weight of the joint to each segment that is connected by this joint. This was achieved by creating new assemblies and assigning half the density to the joints.

After selecting the materials, I proceeded to subtract the above-mentioned parameters of individual segments.

The mechanism uses bearings for smooth running of joints. These are known for their nonlinear friction, which had to be taken into account when creating a mathematical model. I used a Simulink block "Coulomb Viscous friction" to implement the bearing behavior. This block is described by Coulomb and viscous friction values, that had to be identified. I chose to identify them from experimental data. The experiment consisted in bringing the sine wave to the input moments. The amplitude of the torque was chosen so that the leg would move in the range of motion in which leg would typically work.

I identified the coefficients of friction from these experimental MATLAB data with the `fminsearch` function, which optimizes in terms of least squares. The figure 5.3 shows the response of the segment 2 in the simulation and in reality after the identification of these coefficients.

5.3 Linearization of a mathematical model

The mathematical model 5.7 is nonlinear and it is necessary to linearize it for the design of full state feedback linear control. First I made a choice of state variables. These came from reducing the order of derivation method and state variables are independent variables from the mathematical model and their derivatives. Thus state variables are noted: q_1 - angle of rotation of segment 1, $\dot{q_1}$- angular velocity of segment 1, q_2 - angle of rotation of segment 2, $\dot{q_2}$ - angular velocity of segment 2

$$\mathbf{x} = [q_1 \ \dot{q_1} \ q_2 \ \dot{q_2}]^{\mathrm{T}}. \tag{5.8}$$

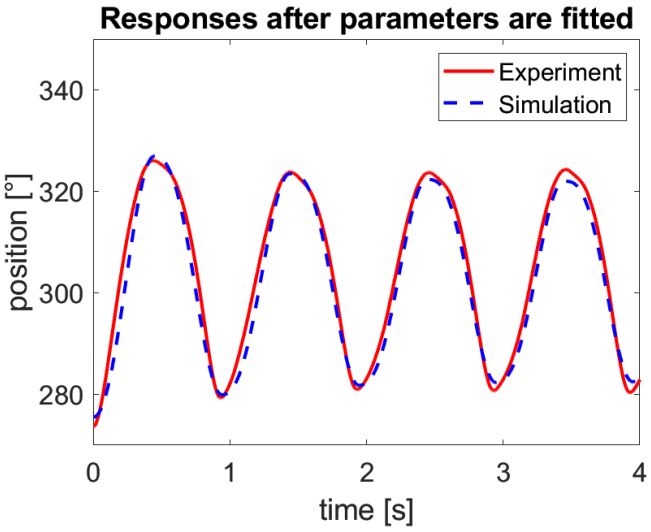


FIGURE 5.3: Simulation and experiment responses after identification of friction parameters

In order to obtain system dynamics given by state-space model

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \end{aligned} \tag{5.9}$$

where $\mathbf{x}$ are state variables, $\mathbf{y}$ are output variables, $\mathbf{u}$ are manipulated variables, $\mathbf{A}$ is state matrix, $\mathbf{B}$ is input matrix, $\mathbf{C}$ is output matrix and $\mathbf{D}$ is feedforward matrix, I continued to linearize the model 5.7 in the sense of:

$$\begin{aligned} \mathbf{J}_{\mathbf{x}} &= \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0,\mathbf{u}_0} = \mathbf{A} \\ \mathbf{J}_{\mathbf{u}} &= \left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right|_{\mathbf{x}_0,\mathbf{u}_0} = \mathbf{B}, \end{aligned} \tag{5.10}$$

where $\mathbf{f}$ are own equations of motion in 5.7.

For this particular leg, matrices A and B have following values

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ -66.12 & -31.84 & 0 & 0.65 \\ 0 & 0 & 0 & 1 \\ 0 & 0.62 & -65.78 & 30.22 \end{bmatrix} \\ \mathbf{B} &= \begin{bmatrix} 0 & 0 \\ 324.85 & 0 \\ 0 & 0 \\ 0 & 308.35 \end{bmatrix} \end{aligned} \tag{5.11}$$

Linearization has been performed using a symbolic toolbox in the MATLAB environment. First, I differentiated my own equations of motion **f** according to the state variables **x** and inserted into the equations the values for the balance of the operating point $\mathbf{x}_0$ and $\mathbf{u}_0$. Operating point has been chosen as a natural resting position of leg - this means 267° for segment 1 (q_1) and 273° for segment 2 (q_2). Velocities ($\dot{q_1}$ and $\dot{q_2}$) are equal to zero and input torques τ_1 and τ_2 are calculated for this equilibrium. Subsequently, I performed a simulation comparison of a nonlinear and a linear model - they behaved almost identically around the operating point.

5.4 Proposed control algorithm

After the linearization of the mathematical model, it was time to design a control algorithm. I decided to use full state feedback, as it provides such system control that the system has prescribed dynamics - theoretically any.

I achieved this with the command `place`, which determines the feedback gain **K** from the prescribed roots. This command has been chosen because it can be used on MIMO systems (which this system is) as opposed to it's alternative - the `acker` command.

The overall scheme of the control circuit with state feedback and observer can be seen in figure 5.4. The vector **r** represents the required value of the state variables **x** here, the vector **u** are then manipulated variables or action interventions - in my case external moments.

State feedback is inherently working with all state variables, but I only measure some of them. In fact, I do not measure all state variables. I only measure angles q_1 and q_2, but no longer their angular velocity. State feedback **K** needs all state variables, so I designed a state observer, which is marked in the diagram with the notation "^". In order for the observer's state variables to converge to the actual state variables, the observer contains feedback gain **L** on a difference of the measurable state variables **y** and their theoretical twins $\hat{\mathbf{y}}$. The gain of this feedback was chosen so that the dynamics of convergence was 6 times faster than the state feedback of the whole system.

5.5 Experimental validation of proposed control

Implemented full state feedback can be seen in the figure 5.5. I used this model for generating vectors of desired torques τ_1 and τ_2, which were subsequently used as feed forward torque command for real leg with custom Python script.

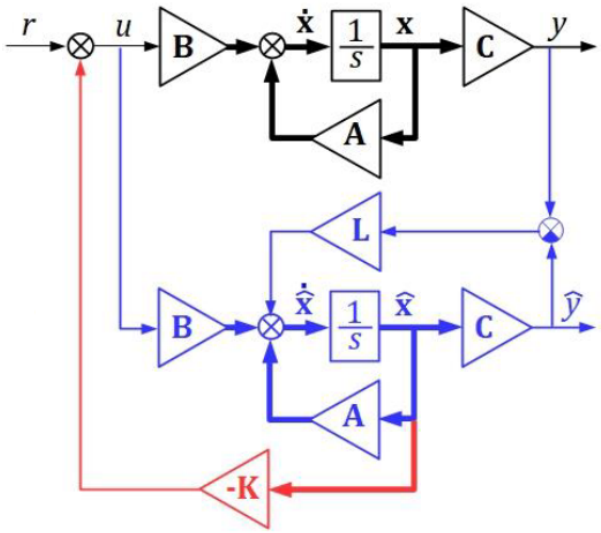


FIGURE 5.4: Closed loop control system block diagram with observer (Vyhlídal, Vrána, and Bušek, 2019)

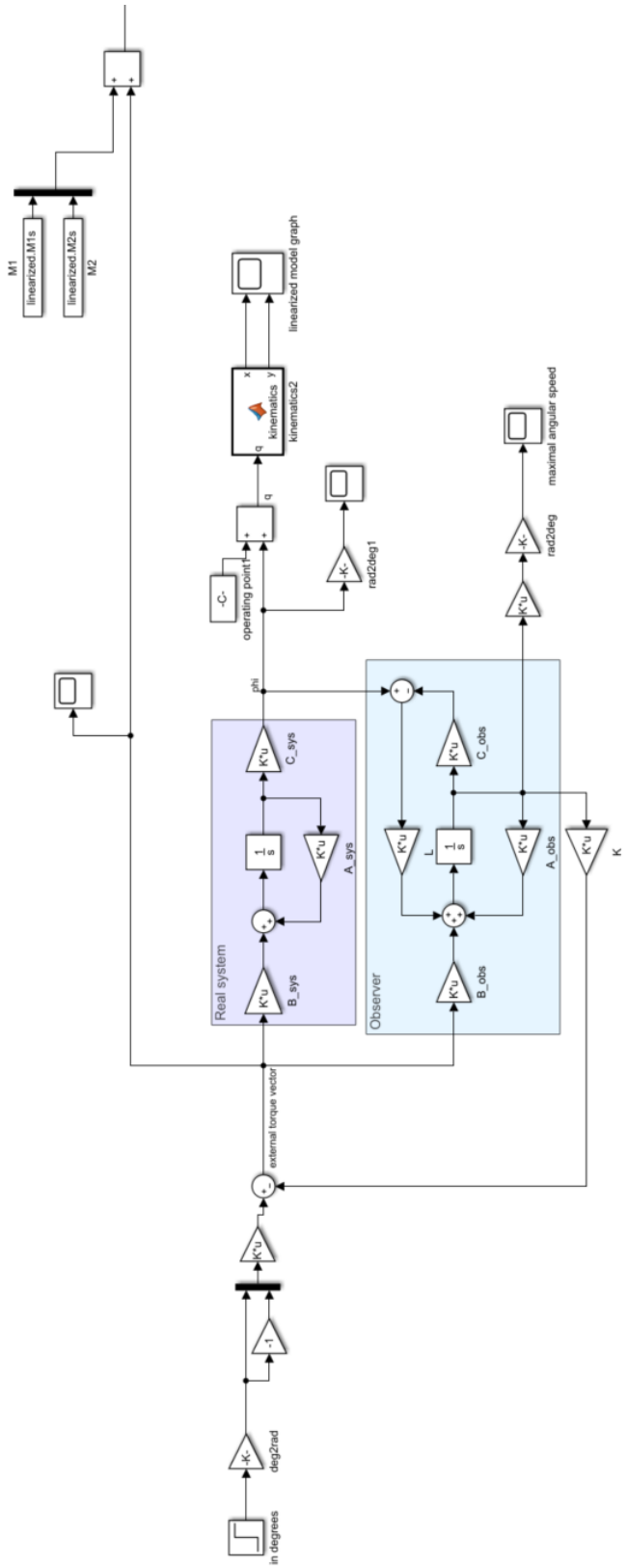


FIGURE 5.5: Closed loop control system block diagram with observer

To verify the correctness of the identified model parameters, I chose the step response with closed-loop poles at $[-7 - 7.007 - 7.014 - 7.021]$. In Figure 5.6 we can see the response of segment 2 from the experiment (red, solid line) and the response from the simulation (blue, dashed line) for desired setpoint 20°. In Figure 5.7, the detail of the run-up from both the experiment and the simulation can then be seen.

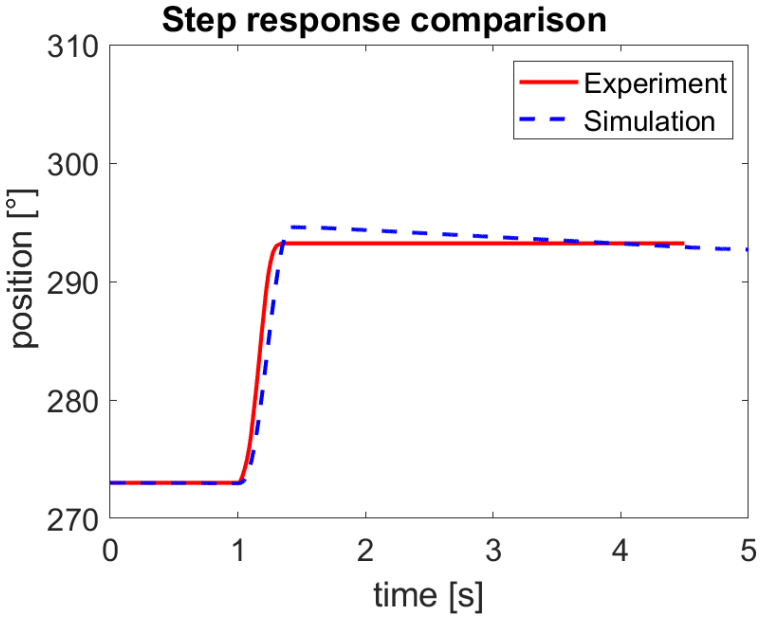


FIGURE 5.6: Comparison of step responses of the experiment and the nonlinear mathematical model

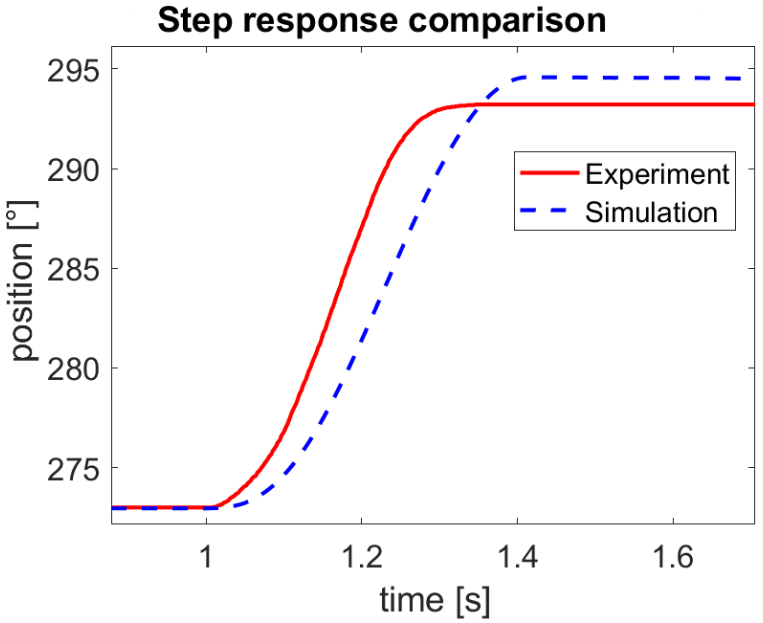


FIGURE 5.7: Comparison of step responses of the experiment and the nonlinear mathematical model - detail

You can see from the detail that the real mechanism has a faster response. Its steepness corresponds to the steeper waveform in Figure 5.3 and is affected by the friction

coefficients. Both the real leg and its nonlinear model have a skew over the required 290°. However, the difference in responses varies with the prescribed roots for the state feedback and the magnitude of the step change in the control variable. All these changes are manifested with the change in the velocity waveform of the driven segments. It can be inferred that this difference is due to the effect of the bearings, which - as already mentioned - are replaced by a single aggregate element. There are 12 bearings per leg, which operate at different speeds. Furthermore, the friction coefficients - as Ruderman and Iwasaki state (Ruderman and Iwasaki, 2015) - depend on the manufacturing roughness, temperature and sliding conditions, which are completely individual for each bearing.

Chapter 6

Proposed control from experimental setting of PID regulators

Alternatively to design control algorithm based on mathematical model, as described in chapter 5, this chapter aims to adjust dynamics of leg's end effector using cascade control with PID regulators. As is known, mathematical models of nontrivial mechanisms can be hard to obtain and it is often necessary to simplify such models in order successfully simulate them (for example using Simulink) and/or design a control algorithm. This turned out to be true for this specific mechanism as we could see in chapter before. Therefore I propose alternative control strategy based on experimental data. Inspired by nature, where for example a cat swing with its body after hitting a hard mat, I decided to investigate only overshoot responses in this chapter.

This chapter first proves that it is possible to change the dynamics of the BLDC motor using ODrive cascade PID controllers. Describes the experimental determination of motor dynamics parameters for individual controller settings. Subsequently, experimental verification of the selection of PID controller parameters for preselected dynamic behavior is done.

6.1 Replacement with second-order system

If it were possible to control the torque of motors with any time course, the end effector could theoretically have any dynamics. In reality, however, this theory is limited by the speed of control and power electronics. For this reason, I chose the prescription of the second-order system, the response of which will be copied by the effector.

Therefore, I implemented the following replacement scheme. The entire four-arm mechanism is replaced by a virtual leg, which begins in the axis of the driven shafts and ends in the effector of the lower segment. It also consists of a spring and a damper, which give it the behavior of a second-order system. The whole diagram can be seen in the figure 6.1.

This method of robot control design with replacement scheme is widely used. Especially where the robot actively responds to its surroundings. The basics of the so called "impedance control" are thoroughly described by Neville Hogan (Hogan, 1984). At present, such control can be observed applied in collaborative robots (Keppler et al., 2018) or other mobile robots (Hyun et al., 2014).

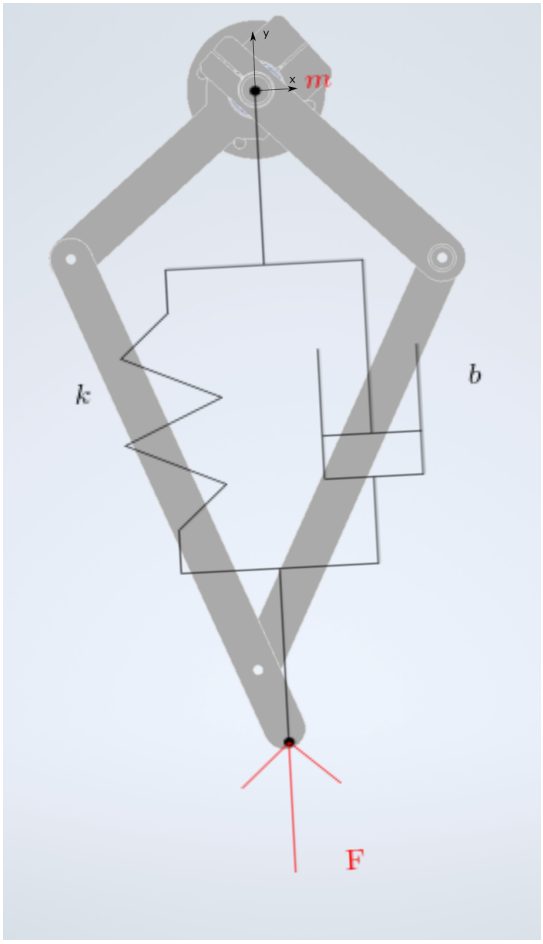


FIGURE 6.1: Diagram of a virtual Leg

Newton's equation of the mechanism in the figure 6.1 is

$$m\ddot{y} = F - ky - b\dot{y}. \tag{6.1}$$

Parameter m is the weight of the leg, b is the coefficient of the damper and k is the coefficient of the spring.

The mathematical rule of the general second order system is expressed by the equation

$$\ddot{y} + 2\xi\omega_n\dot{y} + {\omega_n}^2 y = k_{dc}{\omega_n}^2 u. \tag{6.2}$$

Which can be written as a transfer function in the following form

$$G(s) = \frac{k_{dc}{\omega_n}^2}{s^2 + 2\xi\omega_n s + {\omega_n}^2} = \frac{1}{ms^2 + bs + k}. \tag{6.3}$$

Where parameters m, b and k connect this general mathematical model with my real (i.e. physical) leg. All of them are real positive numbers.

Parameter k_{dc} is a static gain (also called a DC gain), which is ratio of the magnitude of steady-state output y to the magnitude of the step input u. Parameters ξ and ω_n define dynamic behavior of a second-order system.

Parameter ξ is damping ratio. Behaviour of transient response depends on ξ and can be divided into three possibilities. "If $0 < \xi < 1$, the closed-loop poles are complex conjugates and lie in the left-half s plane. The system is then called underdamped, and the transient response is oscillatory. If $\xi = 0$, the transient response does not die out. If $\xi = 1$, the system is called critically damped. Overdamped systems correspond to $\xi > 1$." (Ogata, 2010)

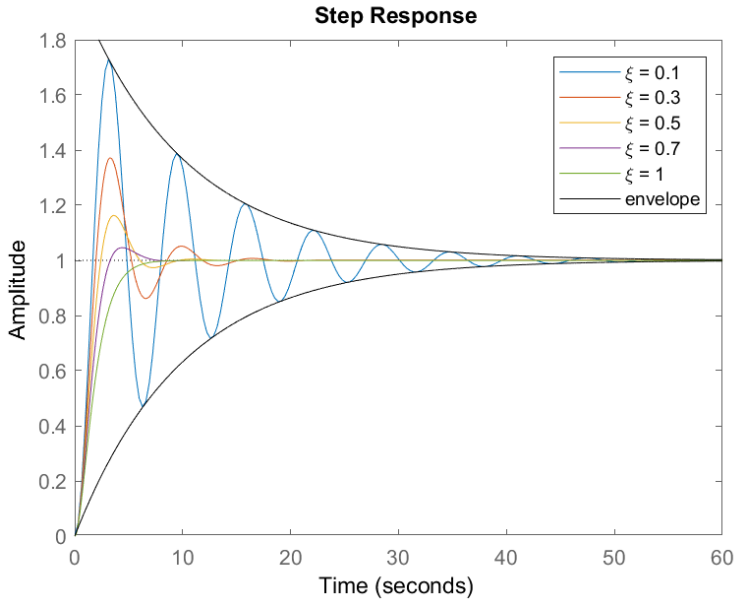


FIGURE 6.2: Response curves for different ξ. $k_{dc} = 1, \omega_n = 1$

This parameter appears in the response of the system as an exponential envelope $e^{-\xi t}$, which the response will never exceed (this can also be seen in figure 6.2 for general second order system). In real systems, damping is most often caused by friction and resistances (e.g. friction in the joints).

Parameter ω_n is the natural frequency (rad/s) at which system oscillates. Responses of systems with different natural frequency ω_n to the step input can be seen in the figure 6.3.

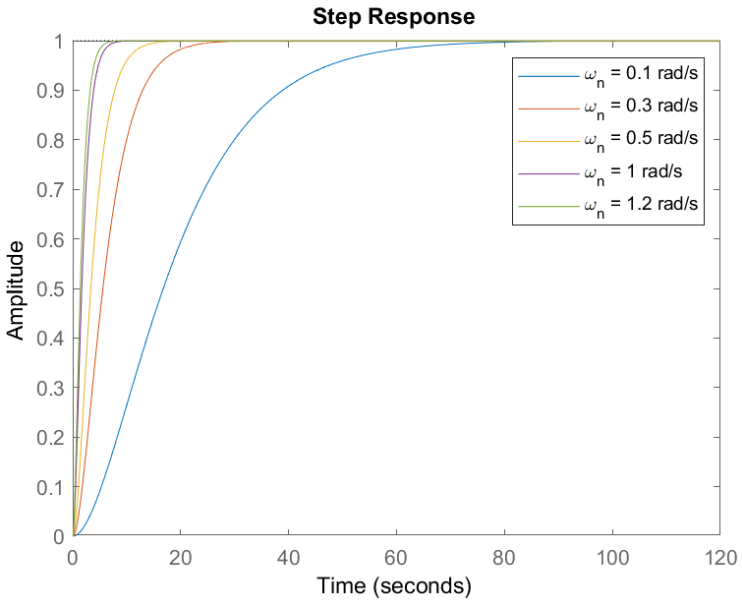


FIGURE 6.3: Response curves for different ω_n. $k_{dc} = 1, \xi = 1$

6.2 Dynamics of BLDC motor with cascade PIDs

This approach uses cascade control used in ODrive board. Cascade control can be understood as several consecutive controllers. Each one adjusts setpoint for following controller. Particular solution implemented in ODrive can be seen in figure 6.4. The outermost is position controller which has only proportional part, it's output is then summed up with velocity feed-forward command. This velocity setpoint goes into velocity controller, that has proportional and integral components. Again, it's output is summed up with current feed-forward command. Last controller is current with proportional and integral components. It is important to note, that PI parameters of current controller are calculated internally in ODrive with respect to individual motor and user should not change them. This left user with 3 parameters to freely change - P component on position and velocity and I component on velocity.

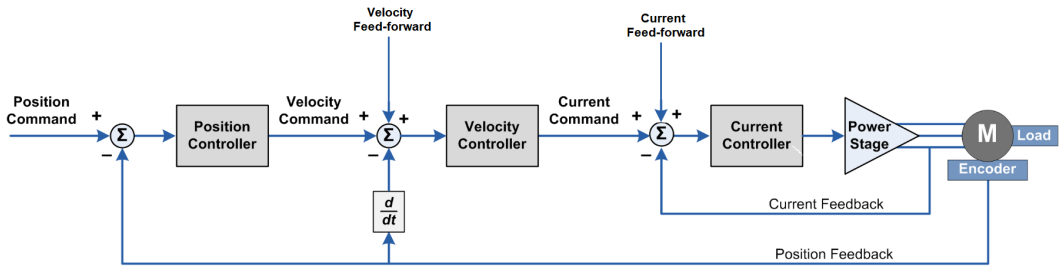


FIGURE 6.4: Cascade control implemented in ODrive (Robotics, 2021a)

Since the BLDC motor is represented by single block in scheme above and naturally, every BLDC motor behaves differently, it is necessary to describe the dynamics of the motor itself. I used a model described by Krishman (Krishnan, 2010), which can be seen in figure 6.5.

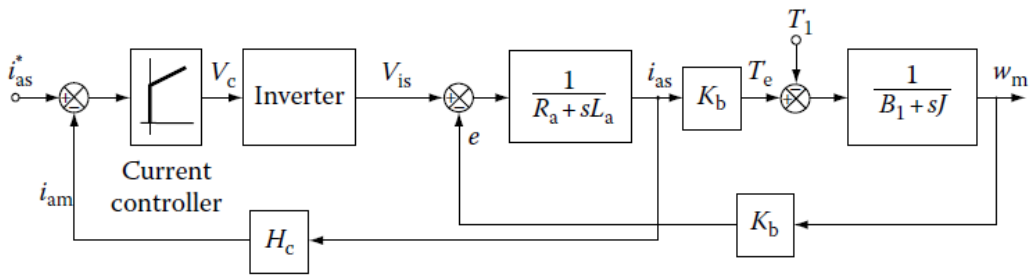


FIGURE 6.5: Current loop of permanent magnet brushless DC motor (Krishnan, 2010)

Where R_a is twice the per phase resistance in PM brushless dc machines, L_a denotes (L-M) in PM brushless DC machines, K_b is induced emf constant, B_1 is friction coefficient, J is total moment of inertia, H_c is gain of the current transducer.

This scheme does not describe the relationship of induced electromagnetic force T_1 and shaft speed ω_m, so it needs to be further described. This relationship is given by

$$T_1 = B_1 \omega_m. \tag{6.4}$$

Taking this dependence into account, scheme of complete inner current loop of BLDC motor can be seen in figure 6.6.

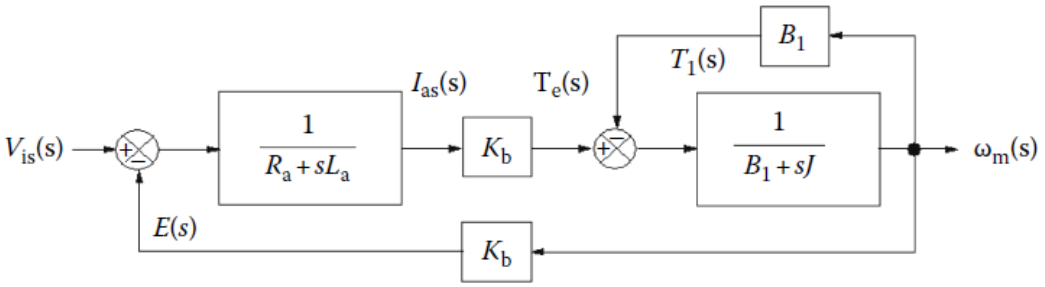


FIGURE 6.6: Inner current loop of permanent magnet brushless DC motor (Krishnan, 2010)

6.3 Changing dynamics of BLDC motor using cascade PIDs

I implemented schemes shown in figures 6.4, 6.5 and 6.6 in Simulink (see figure 6.7) in order to prove, that dynamics of BLDC can be changed by cascade PIDs.

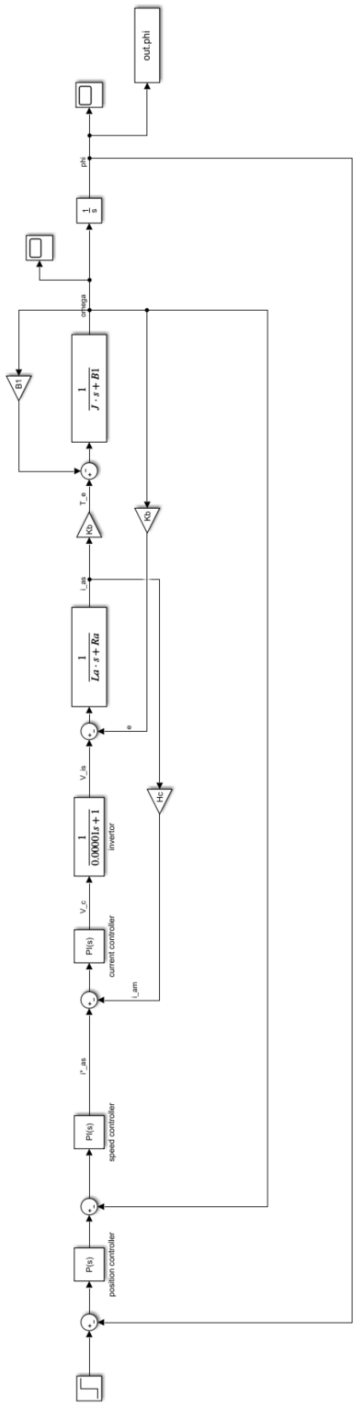


FIGURE 6.7: Closed loop control system block diagram for BLDC

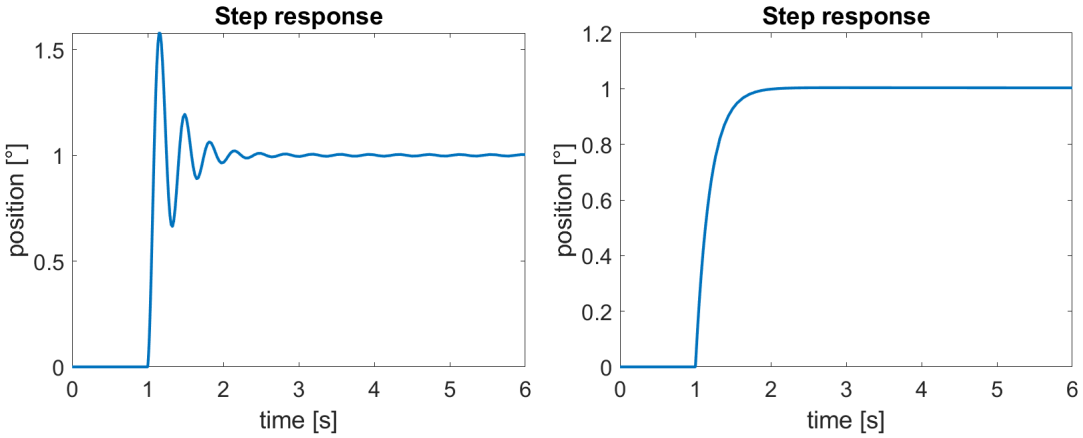
Parameter values used for simulations can be seen in table 6.1.

TABLE 6.1: Parameter values used for BLDC motor simulation

Parameter	Value
R_a	1.2 Ω
L_a	$0.56*10^{-3}$ H
K_t	$25.5*10^{-3}$ NmA^{-1}
t_m	$17.1*10^{-3}$ s
J	$92.5*10^{-7}$ $kg{\cdot}m^2$
p	3 (number of phases)
K_b	0.0764 V/(rad/s)
B_1	0.1 N·m/(rad/s)
H_c	1 V/A

One more thing had to be modeled - Invertor. It has been implemented as first order system with small time constant, as described by Krishman (Krishnan, 2010).

The figure 6.8 shows the step responses for different PID controller settings. We can see underdamped response in figure 6.8a with following PID values: proportional term on position = 100, proportional term on speed = 0.1 and integral term on speed = 10. On the other hand we can see overdamped response in the figure 6.8b with PID values: proportional term on position = 8, proportional term on speed = 5 and integral term on speed = 0.3. It is clear that we can change dynamics of this closed loop with BLDC using position and speed PID controllers.



(A) Response for first set of parameters

(B) Response for second set of parameters

FIGURE 6.8: Step responses for different PID parameters

6.4 Experimental identification of dynamics parameters

The main idea of this method is to obtain possible combinations of damping and frequency of the second order system depending on the settings of the PID controllers from experimental data. From these, subsequently adjust the PID parameters according to the desired dynamic properties.

First, I created a Python script that sets the BLDC motor controllers and measures the step response for each selected combination of PID parameters. In this way, I went through each combination of PID parameters until the leg became unstable. Then I continued in the analytical phase.

One such response can be seen in the figure 6.9. Specifically, the PID controllers had the following values: proportional term on position = 50, proportional term on speed = 0.046 and integral term on speed = 0.001. To illustrate the dynamic response options, I chose a response with an overshoot.

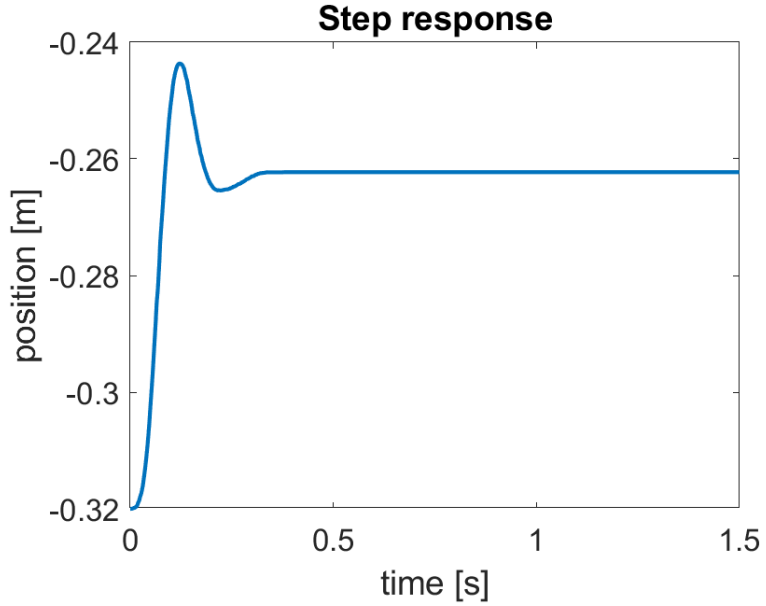


FIGURE 6.9: Step response of real mechanism for one combination of PID parameters

Only one variable was used for the following analysis - the y position of the end effector from figure 5.1. This was calculated by a simple goniometric calculation from the angles q_1 and q_2 (equation 6.5). I omitted the x coordinates because the effector moved symmetrically along the y axis.

$$y(t) = l_1 \sin(q_1) + l_2 \sin(q_2). \tag{6.5}$$

I then processed the experimental data to find peaks in the data to determine damping and frequency. The data contained noise, so as part of the pre-processing, I smoothed it using the MATLAB function `smooth`. Next, I used the function `findpeaks` to find the peaks and their position in time in the response data. The position of the found peaks for this particular measurement can be seen as red circles in the figure 6.10.

I then identified the damping coefficient ξ of second order system (of which I simplify the system for it's understandable parameters) using the Logrithmic decrement method (Casiano, 2016) to the peak positions thus found. An auxiliary formula is used for this method:

$$\beta = \ln\left(\frac{A_{t_1}}{A_{t_2}}\right), \tag{6.6}$$

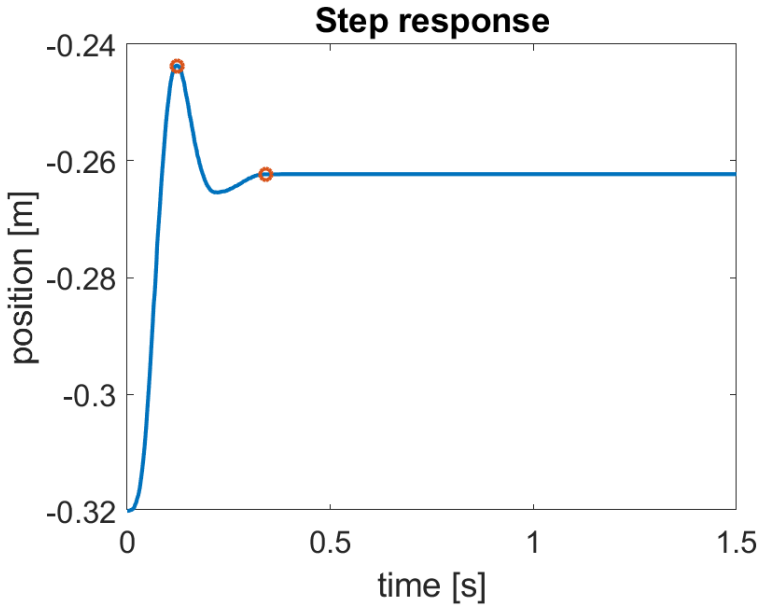


FIGURE 6.10: Peaks in step response of real mechanism for one combination of PID parameters

where A_{t_1} and A_{t_2} are measured amplitude envelope values at t_1 and t_2. Damping is then calculated as:

$$\xi = \frac{\beta}{(\sqrt{4\pi^2 + \beta^2})}. \tag{6.7}$$

The natural frequency of a particular response was calculated using:

$$\omega_n[\text{Hz}] = \frac{1}{(t_2 - t_1)}. \tag{6.8}$$

Investigating only overshoot responses, I discarded measurement that did not contained at least 2 peaks. For simplification, I declared measurements with single peak as critically damped ($\xi = 1$) and measurements without any peak at all as damped with $\xi = 2$.

I then obtained pairs of 4D data from all measurements. Each pair had "axes": a P position component, a P velocity component and an I velocity component. The values of the pairs were then set to the parameters ξ and ω_n. Since I had not measured all possible combinations, these "NaNs" needed to be completed. MATLAB does not have its own function to fill in the 4D array, so I filled in the missing ones using the custom function `inpaintn` (Garcia, 2020). I made measurements at discrete points, cutting off the set of possible combinations of ξ and ω_n. Therefore, I filled this space at least partially with cubic interpolation using the `interp3` command. How cubic interpolation works for 1D and 2D signal can be seen in the figure 6.11.

The post-processing of the data followed. In some cases, either damping or frequency points have been extrapolated to negative values. This is not physically possible and therefore I deleted these points and visualized the rest. A graphical representation of the possible combinations of ξ and ω_n can be seen in the figure 6.12.

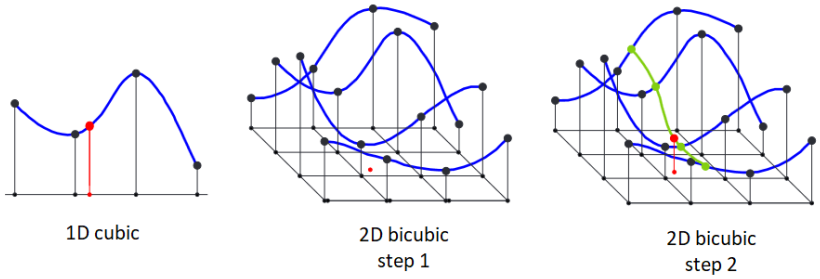


FIGURE 6.11: 1D and 2D cubic interpolation (Ing. Václav Hlaváč, 2020, translated)

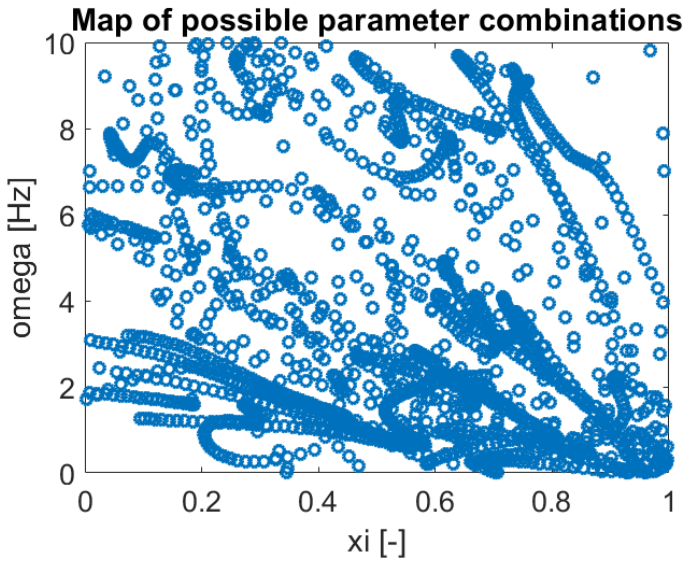


FIGURE 6.12: 2D map of possible parameter combinations

These are combinations that are theoretically realistic according to experimental data and their extrapolation. Now I just need to select desired ξ and ω_n values and the MATLAB script will return the necessary PID controller parameters.

In case the desired values do not correspond directly to one of the points, the nearest neighbor is found using the function `dsearchn`. Interpolation cannot be used here a priori, because it is not certain that the neighboring points in the graph 6.12 correspond to the "neighboring combinations" of PID parameters.

6.5 Experimental validation of proposed control

In this section I will verify the proposed control using PIDs controllers. This consists of selecting the desired dynamic properties, finding the appropriate PID settings and experimentally testing the step response.

First, I chose the desired parameters as follows: $\xi = 0.24$ and $\omega_n = 5$. Then, I found the exact indexes of desired ξ value in extrapolated 4D matrix using the function `ind2sub` and then corresponding PID settings. Subsequently, I set these values to ODrive board and measured the step response, which can be seen in the figure 6.13.

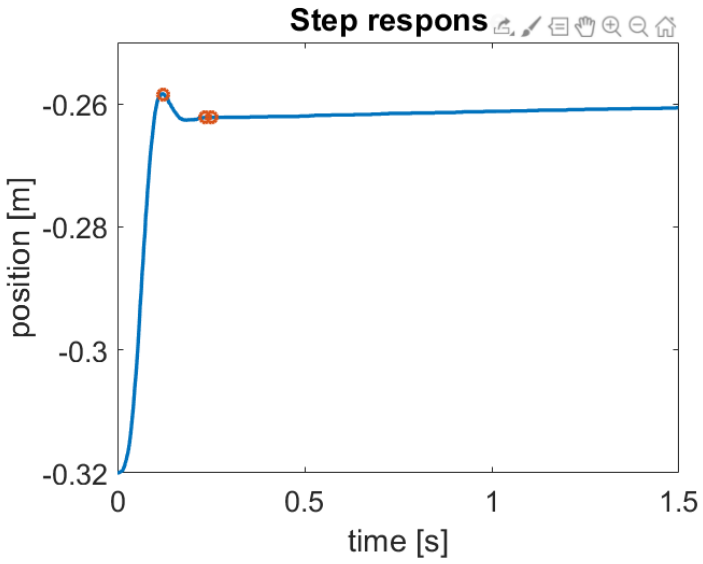


FIGURE 6.13: Step response for PID method validation

Finally, I needed to identify the dynamics parameters. For which I used the same method as in section 6.4. And they were: $\xi = 0.229$, $\omega_n = 5.06$. Both values lie in 5% interval around the desired points. Given the fact, that I describe the entire closed loop in figure 6.7 with second order, I consider this method as successful.

Chapter 7

Software implementation

In this section i am going to describe which software I have used and how I have implemented the function described in previous chapters. I will describe main Python scripts in first section and MATLAB scripts in second section.

7.1 ODrive setup

First of all, I will present the way how I have controled the ODrive. I used the ODrive Python library [1], which communicates with ODrive via USB. As mentioned, ODrive can be controller for example from Arduino, but Python is suitable for rapid prototyping and development. Python in version 3.8.5 has been used during whole development.

Library has command line version, which can be started with Anaconda Prompt `odrivetool`. After startup, computer connects to ODrive and returns object "odrv0" which can be used for commands. This also means that multiple ODrives can be controlled simultaneously - in Doggo's case one ODrive per one leg.

From now on, It is possible to get ODrive configuration values and change them. This is illustrated in listing 7.1. This configuration is critical since motor and encoder properties (which have been found in data sheets) are used during calibration routine and limit values on current or velocity can be used as safety measures. This configuration can be done only once, because ODrive save all values when command `odrv0.save_configuration()` is called.

```
odrv0.axis0.motor.config.current_lim = 10.0
odrv0.axis1.motor.config.current_lim = 10.0
odrv0.axis0.controller.config.vel_limit = 2.0
odrv0.axis1.controller.config.vel_limit = 2.0
odrv0.config.brake_resistance = 2.05
odrv0.axis0.motor.config.pole_pairs = 11
odrv0.axis1.motor.config.pole_pairs = 11
odrv0.axis0.motor.config.torque_constant = 8.27/340
odrv0.axis1.motor.config.torque_constant = 8.27/340
odrv0.axis0.motor.config.motor_type = 0
odrv0.axis1.motor.config.motor_type = 0
odrv0.axis0.encoder.config.cpr = 4000
odrv0.axis1.encoder.config.cpr = 4000

odrv0.save_configuration()
```

LISTING 7.1: ODrive configuration setup

[1] https://docs.odriverobotics.com/odrivetool.html

Minimal usable code can be seen in listing 7.2. Lines 1-2 run both motor and encoder calibration for both motors and following lines switch motors to closed loop control mode with velocity as setpoint and set arbitrary setpoint.

```
1  odrv0.axis0.requested_state = AXIS_STATE_FULL_CALIBRATION_SEQUENCE
2  odrv0.axis1.requested_state = AXIS_STATE_FULL_CALIBRATION_SEQUENCE
3
4  odrv0.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
5  odrv0.axis0.controller.config.control_mode =
       CONTROL_MODE_VELOCITY_CONTROL
6  odrv0.axis0.controller.input_vel = 1
7
8  odrv0.axis1.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
9  odrv0.axis1.controller.config.control_mode =
       CONTROL_MODE_VELOCITY_CONTROL
10 odrv0.axis1.controller.input_vel = 1
```

LISTING 7.2: Minimal viable commands to control velocity of each motor

However using the command line is not very practical when multiple commands are called in sequenced. I therefore created Python scripts with imported ODrive library for all applications.

7.2 Control ODrive with Python scripts

Script that has been used many times in chapter 5 is in listing 7.3. As mentioned in that chapter, I exported the torque vector from Simulink to .csv file. Python script then reads the file on line 13, sets both ODrive axis to closed loop control mode and sets torque setpoint to desired values with given sampling and simultaneously saves actual positions of linkage segments to vector on lines 39-54. After that, script shuts motors down and exports segments positions vectors to .csv file which is later read in MATLAB.

```
1  from datetime import datetime
2  import odrive
3  from odrive.enums import *
4  import time
5  import numpy as np
6  import matplotlib.pyplot as plt
7
8  # Find a connected ODrive (this will block until you connect one)
9  print("finding an odrive...")
10 odrv0 = odrive.find_any()
11
12 # Load .csv file with precalculated torque vectors
13 torqueMatrix = np.loadtxt('Torque.csv', delimiter = ',')
14
15 # Define time parameters
16 duration = 4.5          # [s]
17 sampling = 0.002        # [s]
18
19 # Wait for user to press enter, then continue
20 input("Press Enter to continue...")
21 time.sleep(5)
22
23 # Set both axis to closed loop control with torque as setpoint
24 odrv0.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
25 odrv0.axis0.controller.config.control_mode =
       CONTROL_MODE_TORQUE_CONTROL
```

```
26
27 odrv0.axis1.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
28 odrv0.axis1.controller.config.control_mode =
       CONTROL_MODE_TORQUE_CONTROL
29
30 # Define working variables
31 positionSegment1 = []
32 positionSegment4 = []
33 currentVector = []
34 timeVector = []
35 i = 0
36
37 startTime = time.time()
38
39 while True:
40     elapsedTime = time.time() - startTime
41
42     if elapsedTime > duration:
43         break
44
45     i = int(np.floor(elapsedTime*1000/2))
46
47     odrv0.axis1.controller.input_torque = torqueMatrix[i][0] # segment
       1
48     odrv0.axis0.controller.input_torque = torqueMatrix[i][1] # segment
       4
49
50     positionSegment1.append(360/4000/3 * odrv0.axis1.encoder.
       shadow_count)
51     positionSegment4.append(360/4000/3 * odrv0.axis0.encoder.
       shadow_count)
52
53     currentVector.append(odrv0.axis1.motor.current_control.Iq_measured)
54
55     timeVector.append(elapsedTime)
56
57 # Stop both axis
58 odrv0.axis0.controller.input_torque = 0
59 odrv0.axis1.controller.input_torque = 0
60
61 odrv0.axis0.requested_state = AXIS_STATE_IDLE
62 odrv0.axis1.requested_state = AXIS_STATE_IDLE
63
64 # Convert posiiton vectors to numpy arrays
65 positionSegment1 = np.array(positionSegment1)
66 positionSegment4 = np.array(positionSegment4)
67
68 positionSegment1 = positionSegment1 - positionSegment1[0] + 3
69 positionSegment4 = positionSegment4 - positionSegment4[0] - 3
70
71 # Plot motor positon with respect to time
72 plt.figure(0)
73 plt.plot(timeVector, positionSegment1, color='orange', label='Segment1'
       )
74 plt.plot(timeVector, positionSegment4, color='blue', label='Segment4')
75 plt.title("Position")
76 plt.xlabel('time [s]')
77 plt.ylabel('angle [deg]')
78 plt.legend()
79
80 # Load .csv file with simulation
81 positionSimulation = np.loadtxt('simulation.csv', delimiter = ',')
82
```

```
83 ## Compare simulation and exprimental results
84 plt.figure(1)
85 plt.plot(timeVector, positionSegment1, color='orange', label='
       Experiment')
86 plt.plot(positionSimulation[0:int(duration/sampling), 0],
       positionSimulation[0:int(duration/sampling), 1], color='blue',
       label='Simulation')
87 plt.title("Comparation simulation vs. experiment")
88 plt.xlabel('time [s]')
89 plt.ylabel('angle [deg]')
90 plt.legend()
91
92 # Save position measurement to .csv
93 now = datetime.now()
94 outMeasurementMatrix = np.stack((timeVector, positionSegment1,
       positionSegment4), axis=-1)
95 np.savetxt(now.strftime("%Y%m%d_%H%M%S") + '_positions.csv',
       outMeasurementMatrix, fmt='%2.6f', delimiter=',')
96
97 outTorqueMatrix = np.stack((positionSimulation[:, 0], torqueMatrix
       [:,0], torqueMatrix[:,1]), axis=-1)
98 np.savetxt(now.strftime("%Y%m%d_%H%M%S") + '_torque.csv',
       outTorqueMatrix, fmt='%2.6f', delimiter=',')
```

LISTING 7.3: Python script to measure response for given torque vector

The main Python workhorse in chapter 6 is in listing 7.4. I used Numpy library to create vectors of unique combinations on lines 20-36. Combinations have subsequently been used as PID parameters. For each combination, script set the parameters, performed and recorded a step response and saved the response to .csv file, which had custom name containing PID parameters and timestamp.

```
1  from datetime import datetime
2  import odrive
3  from odrive.enums import *
4  import time
5  import numpy as np
6  import matplotlib.pyplot as plt
7
8  # Find a connected ODrive (this will block until you connect one)
9  print("finding an odrive...")
10 odrv0 = odrive.find_any()
11
12 # Define time parameters
13 duration = 1.5          # [s]
14
15 posPVector = np.linspace(20, 50, num=4)
16 velPVector = np.linspace(0.002, 0.18, num=10)
17 velIVector = np.linspace(0.002, 0.08, num=10)
18
19 # Prepare vectors
20 firstCol = np.ones((len(velPVector)*len(velIVector), 1))*posPVector[0]
21 secondCol = np.ones((len(velIVector), 1))*velPVector[0]
22 thirdCol = np.reshape(velIVector, (len(velIVector), 1))
23
24 for i in range(1, len(posPVector)):
25     a = np.ones((len(velPVector)*len(velIVector), 1))*posPVector[i]
26     firstCol = np.vstack((firstCol, a))
27
28 for i in range(1, len(posPVector)*len(velIVector)):
29     a = np.ones((len(velIVector), 1))*velPVector[i%len(velPVector)]
```

```
30      secondCol = np.vstack((secondCol, a))
31
32  for i in range(len(posPVector)*len(velIVector)-1):
33      a = np.reshape(velIVector, (len(velIVector), 1))
34      thirdCol = np.vstack((thirdCol, a))
35
36  allCombinations = np.hstack((firstCol, secondCol, thirdCol))
37
38  # Wait for user to press enter, then continue
39  input("Press Enter to continue...")
40  time.sleep(1)
41
42  positionS1Offset = odrv0.axis1.encoder.pos_estimate
43  positionS4Offset = odrv0.axis0.encoder.pos_estimate
44
45  for i in range(len(allCombinations)):
46      posP = allCombinations[i][0]
47      velP = allCombinations[i][1]
48      velI = allCombinations[i][2]
49
50      print("posP:   {:.3f} velP:   {:.3f} velI:   {:.3f}".format(posP,
        velP, velI))
51
52      odrv0.axis0.controller.config.pos_gain = posP
53      odrv0.axis0.controller.config.vel_gain = velP
54      odrv0.axis0.controller.config.vel_integrator_gain = velI
55
56      odrv0.axis1.controller.config.pos_gain = posP
57      odrv0.axis1.controller.config.vel_gain = velP
58      odrv0.axis1.controller.config.vel_integrator_gain = velI
59
60      # Set both axis to closed loop control with torque as setpoint
61      odrv0.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
62      odrv0.axis0.controller.config.control_mode =
        CONTROL_MODE_POSITION_CONTROL
63
64      odrv0.axis1.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
65      odrv0.axis1.controller.config.control_mode =
        CONTROL_MODE_POSITION_CONTROL
66
67      # Define working variables
68      positionSegment1 = []
69      positionSegment4 = []
70      timeVector = []
71
72      startTime = time.time()
73
74      while True:
75          elapsedTime = time.time() - startTime
76
77          if elapsedTime > duration:
78              break
79
80          odrv0.axis1.controller.input_pos = positionS1Offset + 0.4
81          odrv0.axis0.controller.input_pos = positionS4Offset - 0.4
82
83          positionSegment1.append(360/4000/3 * odrv0.axis1.encoder.
        shadow_count)
84          positionSegment4.append(360/4000/3 * odrv0.axis0.encoder.
        shadow_count)
85
86          timeVector.append(elapsedTime)
87
```

```
88      # Stop both axis
89      odrv0.axis1.controller.input_pos = positionS1Offset
90      odrv0.axis0.controller.input_pos = positionS4Offset
91
92      odrv0.axis0.requested_state = AXIS_STATE_IDLE
93      odrv0.axis1.requested_state = AXIS_STATE_IDLE
94
95      time.sleep(2)
96
97      # Convert position vectors to numpy arrays
98      positionSegment1 = np.array(positionSegment1)
99      positionSegment4 = np.array(positionSegment4)
100
101     positionSegment1 = positionSegment1 - positionSegment1[0]
102     positionSegment4 = positionSegment4 - positionSegment4[0]
103
104     # Save position measurement to .csv
105     now = datetime.now()
106     outMeasurementMatrix = np.stack((timeVector, positionSegment1,
        positionSegment4), axis=-1)
107     np.savetxt(now.strftime("%Y%m%d_%H%M%S") + "_{:.3f};{:.3f};{:.3f}".
        format(posP, velP, velI) + '_positions.csv', outMeasurementMatrix,
        fmt='%2.6f', delimiter=',')
```

LISTING 7.4: Python script to measure responses for different PID parameters

7.3 MATLAB scripts

In this section, I will comment all necessary information about SW implementation in MATLAB.

All simulations for chapters 5 and 6 are initialized from MATLAB scripts, namely their parameters. When multiple constants are related to each other, I used structure array. This maintains the code clean and also simplifies calling function with many parameters, because now the function has only one parameter - this structure.

Mentioned function can be found in listing 7.5. This function is used to linearizes the mathematical model around a operating point. Function uses symbolic toolbox for derivation which helps to prevent errors in typed equations. Symbolic variables are defined on lines 13-19 and dynamics equations from chapter 5 are on lines 21-44. The function returns structure, which contains torques for equilibrium (lines 45-46), state space matrices (lines 49-93) and full state feedback gain and observer gain (lines 107-113). I had to implement differentiation with respect to another function - this can be seen on lines 119-127.

```
1  function [linearized] = linearize(params, operatingPoint, poles)
2  % Liarize decoupled fivebar mechanism
3
4  % Parse input
5  % Operating point
6  q1s = operatingPoint.q1s;
7  dq1s = operatingPoint.dq1s;
8
9  q2s = operatingPoint.q2s;
10 dq2s = operatingPoint.dq2s;
11
12 %% Symbolic variables
13 syms q1(t) q2(t) t M1 M2
```

```
14
15 dq1 = diff(q1);
16 ddq1 = diff(dq1);
17
18 dq2 = diff(q2);
19 ddq2 = diff(dq2);
20
21 %% Dynamic equations
22 % inertia matrix
23 d11 = params.m1*params.lc1^2 + params.m3*params.lc3^2 + params.m4*
       params.l1^2 + params.I1S1 + params.I3S3;
24 d22 = params.m2*params.lc2^2 + params.m3*params.l2^2 + params.m4*params
       .lc4^2 + params.I2S2 + params.I4S4;
25 d12 = (params.m3*params.l2*params.lc3 - params.m4*params.l1*params.lc4)
       *cos(q2-q1);
26 d21 = d12;
27
28 % friction matrix
29 c11 = params.frictionc11;
30 c12 = params.frictionc12;
31 c21 = params.frictionc21;
32 c22 = params.frictionc22;
33 C1 = [c11 c12;
34       c21 c22];
35
36 % gravitational vector
37 g1 = params.g*cos(q1)*(params.m1*params.lc1 + params.m3*params.lc3 +
       params.m4*params.l1);
38 g2 = params.g*cos(q2)*(params.m2*params.lc2 + params.m3*params.l2 +
       params.m4*params.lc4);
39
40 G = [g1; g2];
41
42 % equations
43 eq1 = 1/d11*(M1 - c11*dq1 - c12*dq2 - g1);
44 eq2 = 1/d22*(M2 - c21*dq1 - c22*dq2 - g2);
45
46 linearized.M1s = c11*dq2s + c12*dq1s + double(subs(g1, q1, q1s));
47 linearized.M2s = c21*dq2s + c22*dq1s + double(subs(g2, q2, q2s));
48
49 %% Differentiate
50 % Jacobian
51 J11_lin = diff2(eq1, q1);
52 J11_lin = subs(J11_lin, q1, q1s);
53 J12_lin = diff2(eq1, dq1);
54 J13_lin = diff2(eq1, q2);
55 J14_lin = diff2(eq1, dq2);
56
57 J21_lin = diff2(eq2, q1);
58 J22_lin = diff2(eq2, dq1);
59 J23_lin = diff2(eq2, q2);
60 J23_lin = subs(J23_lin, q2, q2s);
61 J24_lin = diff2(eq2, dq2);
62
63 % Convert to double
64 J11_lin = double(J11_lin);
65 J12_lin = double(J12_lin);
66 J13_lin = double(J13_lin);
67 J14_lin = double(J14_lin);
68
69 J21_lin = double(J21_lin);
70 J22_lin = double(J22_lin);
71 J23_lin = double(J23_lin);
```

```
72 J24_lin = double(J24_lin);
73
74 % B
75 B21_lin = double(diff2(eq1, M1));
76 B42_lin = double(diff2(eq2, M2));
77
78 %% State space representation
79
80 linearized.A = [0   1   0   0;
81                 J11_lin J12_lin J13_lin J14_lin;
82                 0    0   0   1;
83                 J21_lin J22_lin J23_lin J24_lin];
84
85 linearized.B = [0   0;
86                  B21_lin 0;
87                  0   0;
88                  0   B42_lin];
89
90 linearized.C = [1   0   0   0;
91                  0   0   1   0];
92
93 linearized.D = [0; 0];
94
95 %% Controllability, observability, poleplacement, observer
96
97 % disp('The controllability rank on segment 1 is:')
98 % disp(rank(ctrb(linearized.A1, linearized.B1)))
99 % disp('The observability rank on segment 1 is:')
100 % disp(rank(obsv(linearized.A1, linearized.C1)))
101 %
102 % disp('The controllability rank on segment 4 is:')
103 % disp(rank(ctrb(linearized.A2, linearized.B2)))
104 % disp('The observability rank on segment 4 is:')
105 % disp(rank(obsv(linearized.A2, linearized.C2)))
106
107 %% Full state feedback
108 P = poles.P;
109 linearized.K = place(linearized.A, linearized.B, P);
110
111 % Observer
112 Po = poles.Po;
113 linearized.L = (place(linearized.A',linearized.C',Po))';
114
115 % Calculate setpoint compensation
116 linearized.Nbar = inv(-linearized.C*inv(linearized.A-linearized.B*
    linearized.K)*linearized.B+linearized.D);
117 end
118
119 %% Function for differentiation
120 function [f_out] = diff2(f_in,ableiten)
121 %src: https://www.mathworks.com/matlabcentral/answers/159812-symbolic-
    math-toolbox-derive-a-function-with-respect-to-another-function
122 syms substitute;
123 f_zwischen=subs(f_in,ableiten,substitute);
124 f_zwischen_diff=diff(f_zwischen,substitute);
125 f_out=subs(f_zwischen_diff,substitute,ableiten);
126
127 end
```

LISTING 7.5: Function for linearization

Following script 7.6 has been used in chapter 6. It's job is to load all .csv files in

given folder with measured responses and for each file parse PID parameters from it's name and call custom function "identifyParams" which returns dynamics parameters ξ and ω_n. When all .csv files are processed, vectors containing PID values and dynamics parameters are stored in .mat files for later use.

```
clc
clear all
close all

% Get all files
myFiles = dir('C:\Users\Frantisek\OneDrive - esk vysok u en technick v Praze\Diplomov pr ce_git\aclab_quadruped\[SIM] Simulations\Lagrange\Spong');
% Get the filenames
filenames = {myFiles(:).name}';
% Get only csv files
csvFiles = filenames(endsWith(filenames,".csv"));
% Filter out measurements csv files
csvFiles = csvFiles(contains(csvFiles, "positions"));

% Initialize vectors to hold values
posPVector = [];
velPVector = [];
velIVector = [];
wnVector = [];
xiVector = [];

saveMilestone = 50;
%% For all csv files
for i = 1:length(csvFiles)
    singleFile = string(csvFiles(i));
    % Parse filename for PID parameters
    nameParsed = split(singleFile,{';', '_'});
    posP = str2double(nameParsed(3));
    velP = str2double(nameParsed(4));
    velI = str2double(nameParsed(5));

    % Obtain second order system parameters from external function
    [wn, xi] = identifyParams(singleFile, false);

    % Store data to vectors
    posPVector = [posPVector; posP];
    velPVector = [velPVector; velP];
    velIVector = [velIVector; velI];

    wnVector = [wnVector; wn];
    xiVector = [xiVector; xi];

    % Save after every 50 files
    if i > saveMilestone
        save('identifiedParameters','posPVector','posPVector','velIVector','wnVector','xiVector')
        saveMilestone = saveMilestone + 50;
    end
end

save('identifiedParameters','posPVector','velPVector','velIVector','wnVector','xiVector')
```

LISTING 7.6: Step responses processing

Function "identifyParams" is listed in 7.7. Is has two parameters. First is name of .csv file and the second one is boolean value whether to plot analyzed data. The function opens and loads .csv file to vectors, which are then smoothed and analyzed using the logarithmic decrement method.

```
function [wn, xi] = identifyParams(filename, doPlot)
%% Load csv file
clc
close all

opts = delimitedTextImportOptions("NumVariables", 3);

% Specify range and delimiter
opts.DataLines = [1, Inf];
opts.Delimiter = ",";

% Specify column names and types
opts.VariableNames = ["time", "angle1", "angle2"];
opts.VariableTypes = ["double", "double", "double"];

% Specify file level properties
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";

% Import the data
measurement = readtable(filename, opts);
% Clear temporary variables
clear opts

%% Parameters identification
% Parse data
t = measurement.time;
y = measurement.angle1;
y = 0.16*cos(deg2rad(measurement.angle1))+0.16*cos(deg2rad(measurement.
    angle2));

%% Logrithmic decrement method
[pks,locs] = findpeaks(smooth(-y), t); % Signal processing toolbox

locs

if isempty(pks)
    xi = 2;
    wn = 0;
elseif size(pks) == 1
    xi = 1;
    wn = 0;
else
    delta = -log(pks(1)/pks(2));
    xi = delta/sqrt((2*pi)^2+delta^2);
    wn = 1/(locs(2)-locs(1));
end

%% Plot graphs
if doPlot
    invertedY = smooth(-y); %smooth(max(y) - y);
    [pks,locs] = findpeaks(invertedY, t); % signal processing toolbox

    plot(t, -y, 'LineWidth', 2)
    title("Step response")
    xlabel("time [s]")
    ylabel("position [m]")
```

```
57      set(gca,'FontSize',16)
58      hold on
59      plot(locs, pks,'o', 'LineWidth', 2)
60  end
61  end
```

LISTING 7.7: Function "identifyParams"

Saved .mat file in 7.6 is then used in following listing 7.8. Program loads the data from .mat file into matrices (lines 5-19), fills them with the `inpaintn` method (lines 25-26), interpolate the data for finer sampling with `interp3` function and plot all possible variants of dynamics parameters. At the end of the code (lines 56-70), it takes desired dynamics parameters, finds nearest theoretically possible one and returns it's PID prameters.

```
1  clc
2  close all
3  clear all
4
5  load identifiedParameters.mat
6
7  x = unique(posPVector);
8  y = unique(velPVector);
9  z = unique(velIVector);
10 omegaMatrix = NaN(length(x),length(y),length(z));
11 xiMatrix = NaN(length(x),length(y),length(z));
12
13 for i = 1:length(posPVector)
14     x_ind = find(x == posPVector(i));
15     y_ind = find(y == velPVector(i));
16     z_ind = find(z == velIVector(i));
17     omegaMatrix(x_ind,y_ind,z_ind) = wnVector(i);
18     xiMatrix(x_ind,y_ind,z_ind) = xiVector(i);
19 end
20
21 % How much "sparse" is this matrix
22 % sum(isnan(data),'all');
23 % numel(data);
24
25 % Fill NaNs
26 omegaMatrix = inpaintn(omegaMatrix);
27 xiMatrix = inpaintn(xiMatrix);
28
29 % Interpolate for a new meshgrid
30 [Xq,Yq,Zq] = meshgrid(x(1):1:x(end), y(1):0.02:y(end), z(1):0.02:z(end)
       );
31 omegaInterpolated = interp3(y, x, z, omegaMatrix, Yq, Xq, Zq, 'cubic');
32 xiInterpolated = interp3(y, x, z, xiMatrix, Yq, Xq, Zq, 'cubic');
33
34 % Flatten for plotting
35 omegaFlattened = reshape(omegaInterpolated, [], 1);
36 xiFlattened = reshape(xiInterpolated, [], 1);
37
38 % Remove negative frequencies and damping
39 omegaFlattened(xiFlattened<0) = NaN;
40 xiFlattened(xiFlattened<0) = NaN;
41
42 xiFlattened(omegaFlattened<0) = NaN;
43 omegaFlattened(omegaFlattened<0) = NaN;
44
45 figure
46 plot(xiFlattened, omegaFlattened, 'o', 'LineWidth', 2)
```

```
47 title("Map of possible parameter combinations")
48 xlabel("xi [-]")
49 ylabel("omega [Hz]")
50 xlim([0 1])
51 ylim([0 10])
52 hold on
53 set(gca,'FontSize',16)
54
55 % transform to 2D
56 combinedMatrix = [xiFlattened omegaFlattened];
57 desiredPoint = [0.23, 5];
58 [k,dist] = dsearchn(combinedMatrix,desiredPoint);
59 plot(combinedMatrix(k, 1), combinedMatrix(k, 2), '*', 'LineWidth', 2)
60 %%
61 closestValueXi = xiFlattened(k);
62 closestValueOmega = omegaFlattened(k);
63
64 %result = find(X==5);
65 [r,c,v] = ind2sub(size(omegaInterpolated),find(omegaInterpolated ==
       closestValueOmega)); % src: https://uk.mathworks.com/matlabcentral/
       answers/789-using-find-in-a-3d-matrix-in-matlab
66
67 % Desired PID parameters
68 pSpeed = Yq(r)
69 pPosition = Xq(c)
70 iSpeed = Zq(v)
```

LISTING 7.8: Data mining

Chapter 8

Conclusion and future directions

The aim of this master thesis was the realization of a robotic leg and the design of its control. In the chapter 2, I described the evolution of mobile robotics and did a review of similar projects and approaches to dynamics management. I then presented in detail the Doggo platform, which I based my development on. Both the mechanical (chapter 3) and eletrical (chapter 4) aspects of the project were discussed. I also listed the changes to some parts that had to be made for the leg to work properly. I described the main electrical components that were used.

In the following chapters, I proposed two approaches to control the dynamics of the leg. The first approach (chapter 5) was based on the mathematical model of the fivebar linkage. I supplemented this with a damping matrix and identified all the necessary parameters. In validating this method, there were noticeable deviations between simulation and experiment. These deviations are probably due to an oversimplification of the modeling of the applied friction. The second method (chapter 6) is then based on experimental measurement data. I successively analyzed the dynamic parameters of the leg behavior for different settings of the cascaded PID controllers. I then determined the appropriate PID parameters from these based on the prescribed dynamics parameters and tested this. This method proved to work, but it relies on many measurements and any change in the parameters (for example, loading the robot with cargo) would necessitate doing all the measurements again.

Future development should focus on improving the mathematical model from the chapter, especially the friction modeling. As mentioned, many bearings are used throughout the mechanism, and furthermore, the fact that the mechanism consists of many parts and thus friction can occur due to inaccuracies in geometric tolerances needs to be taken into account. Especially for 3D printed parts, the resulting geometric tolerances are not very precise.

Nevertheless, I hope that the platform I have implemented will serve students in their future research and development. I believe, that one day we will see fully working mobile robot in our department.

Appendix A

Contents of the attached CD

Folder "`CAD model`" contains my version of CAD model for robotic leg. This was discussed in chapter 3.

Two additional folders are inside the folder "`MATLAB and Simulink`". Files inside folder "`Fivebar linkage`" were used for chapter 5. Namely Simulink model of five-bar linkage, it's setup livescript and custom function for linearization. Folder "`BLDC`" contains Simulink model, corresponding setup script used in chapter 6 and script used for data analysis.

The custom Python scripts used to control ODrive are in the folder "`Python scripts`".

Bibliography

3DXR (2021). *T-Motor MN5212 KV340*. URL: https://www.3dxr.co.uk/multirotor-c3/multirotor-motors-c35/t-motor-mn5212-kv340-p3086 (visited on 07/03/2021).

Al-Zughaibi, Ali I. H. (2020). "Nonlinear Friction Model for Passive Suspension System Identification and Effectiveness". In: *Nonlinear Systems*. Ed. by Walter Legnani and Terry E. Moschandreou. Rijeka: IntechOpen. Chap. 12. DOI: 10.5772/intechopen.86055. URL: https://doi.org/10.5772/intechopen.86055.

Blackman, D. J. et al. (2016). "Gait development on Minitaur, a direct drive quadrupedal robot". In: *SPIE Defense + Security*.

Casiano, M.J. (2016). "Extracting Damping Ratio From Dynamic Data and Numerical Solutions". In:

Daily, The Stanford (2021). *Stanford Doggo*. URL: https://www.stanforddaily.com/2019/05/31/stanford-doggo-students-develop-open-source-agile-quadruped-robot/ (visited on 07/04/2021).

Digi-Key (2021). *AS5047D-TS$_E$K$_A$B*. URL: https://www.digikey.cz/product-detail/en/ams/AS5047D-TS-EK-AB/AS5047D-TS-EK-AB-ND/5125788 (visited on 07/03/2021).

Dynamics, Boston (2020). *Do You Love Me?* URL: https://www.youtube.com/watch?v=fn3KWM1kuAw (visited on 08/08/2021).

– (2021). *Spot explorer*. URL: https://www.bostondynamics.com/spot# (visited on 07/04/2021).

Garcia, Damien (2020). *Inpaint over missing data in 1-D, 2-D, 3-D,... ND arrays*. URL: https://www.mathworks.com/matlabcentral/fileexchange/27994-inpaint-over-missing-data-in-1-d-2-d-3-d-nd-arrays (visited on 08/08/2021).

Hobert, Laurens et al. (2015). "Enhancements of V2X communication in support of cooperative autonomous driving". In: *IEEE Communications Magazine* 53.12, pp. 64–70. DOI: 10.1109/MCOM.2015.7355568.

Hogan, N. (1984). "Impedance Control: An Approach to Manipulation". In: *1984 American Control Conference*, pp. 304–313. DOI: 10.23919/ACC.1984.4788393.

Hutter, Marco et al. (2013). "Efficient and Versatile Locomotion With Highly Compliant Legs". In: *IEEE/ASME Transactions on Mechatronics* 18.2, pp. 449–458. DOI: 10.1109/TMECH.2012.2222430.

Hyun, Dong Jin et al. (2014). "High speed trot-running: Implementation of a hierarchical controller using proprioceptive impedance control on the MIT Cheetah". In: *The International Journal of Robotics Research* 33.11, pp. 1417–1445. DOI: 10.1177/0278364914532150. eprint: https://doi.org/10.1177/0278364914532150. URL: https://doi.org/10.1177/0278364914532150.

Ing. Václav Hlaváč, CSc. prof. (2020). *Jasové a geometrické transformace*. URL: https://people.ciirc.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/18BrightGeomTxCz.pdf (visited on 08/08/2021).

Institute, MIT Beaver Works Summer (2018). *MIT BWSI Featuring Prof. Sangbae Kim*. URL: https://www.youtube.com/watch?v=7wNZXoJ8fmI (visited on 08/08/2021).

Katz, Benjamin (2018). *A low cost modular actuator for dynamic robots*. URL: https://dspace.mit.edu/handle/1721.1/118671?show=full (visited on 08/08/2021).

Kau, Nathan (2020). *Bill of materials (BOM)*. URL: https://docs.google.com/spreadsheets/d/1K_G3WL54Q7ngNONFQaJda25RrhjKciRs-QOZ2_rLGuY/edit#gid=0 (visited on 12/18/2020).

Kau, Nathan et al. (May 2019). "Stanford Doggo: An Open-Source, Quasi-Direct-Drive Quadruped". In: pp. 6309–6315. DOI: 10.1109/ICRA.2019.8794436.

Kau, Nathau (2021). *Stanford Doggo: an agile back-flipping robot*. URL: https://www.youtube.com/watch?v=YeUpceVrUfg (visited on 07/04/2021).

Keppler, M. et al. (2018). "Elastic Structure Preserving Impedance (ES)Control for Compliantly Actuated Robots". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5861–5868. DOI: 10.1109/IROS.2018.8593415.

Krishnan, R (2010). *Permanent magnet synchronous and brushless DC motor drives*. Boca Raton: CRC Press/Taylor & Francis. ISBN: 978-0-8247-5384-9.

Marko Bjelonic Prajish K. Sankar, C. Dario Bellicoso Heike Vallery and Marco Hutter (2020). "Rolling in the Deep – Hybrid Locomotion for Wheeled-Legged Robots using Online Trajectory Optimization". In:

News, MIT (2021). *MIT's new mini cheetah robot*. URL: https://news.mit.edu/2019/mit-mini-cheetah-first-four-legged-robot-to-backflip-0304 (visited on 07/04/2021).

Ogata, Katsuhiko (2010). "Modern Control Engineering". In: Prentice Hall PTR.

Park, Hae-Won, Meng Yee Chuah, and Sangbae Kim (2014). "Quadruped bounding control with variable duty cycle via vertical impulse scaling". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3245–3252. DOI: 10.1109/IROS.2014.6943013.

Robotics, ODrive (2021a). *Controller with feed forward*. URL: https://docs.odriverobotics.com/controller_with_ff.png (visited on 06/03/2021).

– (2021b). *ODrive v3.6*. URL: https://eu.odriverobotics.com/shop/odrive-v36 (visited on 07/03/2021).

Ruderman, Michael and Makoto Iwasaki (2015). "Observer of Nonlinear Friction Dynamics for Motion Control". In: *IEEE Transactions on Industrial Electronics* 62.9, pp. 5941–5949. DOI: 10.1109/TIE.2015.2435002.

Sander, D.E. et al. (2016). "Simulation of journal bearing friction in severe mixed lubrication – Validation and effect of surface smoothing due to running-in". In: *Tribology International* 96, pp. 173–183. ISSN: 0301-679X. DOI: https://doi.org/10.1016/j.triboint.2015.12.024. URL: https://www.sciencedirect.com/science/article/pii/S0301679X15005927.

Spong, M.W., S. Hutchinson, and M. Vidyasagar (2005). *Robot Modeling and Control*. Wiley. ISBN: 9780471649908. URL: https://books.google.cz/books?id=P4Y4xQEACAAJ.

Stanford (2019). *Meet Doggo: Stanford's student built, four-legged robot*. URL: https://www.youtube.com/watch?v=2E82o2pP9Jo (visited on 12/15/2020).

Tyma CZ, s.r.o. (2020). *Ozubená tyč 3M-48-200*. URL: https://www.tyma.cz/produkty/3m-48-200-al-slitina/ (visited on 12/30/2020).

Vyhlídal, Tomáš, Stanislav Vrána, and Jaroslav Bušek (2019). *Téma 8: Stavový regulátor pro řízení, pozorovatelnost, pozorovatel stavu*. Faculty of Mechanical Engineering CTU in Prague. URL: https://moodle-vyuka.cvut.cz/course/view.php?id=3005 (visited on 05/29/2020).

Zurich, ETH (2021). *Quadruped robot ANYmal on wheels*. URL: https://spectrum.ieee.org/wheels-are-better-than-feet-for-legged-robots (visited on 07/04/2021).