Control Synthesis for Balancing Robots

Toni Silfver

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 23.5.2018

Supervisor and advisor

Prof. Themistoklis Charalambous

Aalto University
School of Electrical Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Toni Silfver

Title Control Synthesis for Balancing Robots

Degree programme Automation and Electrical Engineering

Major Control, Robotics and Autonomous Systems **Code of major** ELEC3025

Supervisor and advisor Prof. Themistoklis Charalambous

Date 23.5.2018 **Number of pages** 82 **Language** English

Abstract

A balancing robot control system is similar to any unstable dynamical system that can be controlled with a PID. Examples of unstable dynamical systems can be e.g. self-balancing scooter and jet fighter missile guidance applications. The versatility and availability of consumer market microcontroller development platforms and control simulation software gives an opportunity for the independent developer to take on an arbitrary unstable dynamics control design assignment. Hence, a balancing control system was utilized in an Arduino based robot that can stabilize itself despite of perturbation and its intrinsic nature of an inverted pendulum.

In this Thesis, a MinSeg balancing robot control design is carefully studied and reconstructed. First, the Equations of Motion (EOM) of the robot wheels, body and motor are designed, linearized and the Transfer Function (TF) is determined for a model of the robot. Then this model hypothesis is applied to practice by constructing a PID controller that stabilizes the TF. Following a successful implementation of the controller, a form of disturbance is added to the model and the robot control is simulated and field-tested. In conclusion, the PID controller is derived more robust and thus disturbance tolerant by redesigning the control system primarily for the simulated environment and then for the real world.

The implemented solution was to divide the subject into parts that begin from the basics of control and continue to the hands on experiments with the balancing robot. Each component of the mathematical framework was evaluated, prototyped and tested. The meticulous fieldwork resulted in more accurate control parameters that contributed to the final robust design application.

Keywords MinSeg, Balancing, Robot, Control

Aalto University
School of Electrical Engineering

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Toni Silfver

Työn nimi Tasapainottavan järjestelmän robottien säätösuunnittelu

Koulutusohjelma Elektroniikka ja sähkötekniikka

Pääaine Säätötekniikka, robotiikka ja autonomiset järjestelmät **Pääaineen koodi** ELEC3025

Työn valvoja Prof. Themistoklis Charalambous

Työn ohjaaja Prof. Themistoklis Charalambous

Päivämäärä 23.5.2018 **Sivumäärä** 82 **Kieli** Englanti

Tiivistelmä
Tasapainottavan järjestelmän robottien säätötekniikka on samankaltainen muiden epästabiilien dynaamisten systeemien kanssa, joita voidaan ohjata PID-säätimellä. Epästabiileja dynaamisia systeemejä ovat esimerkiksi tasapainoskootteri ja suihkuhävittäjän ohjusohjausjärjestelmä. Kuluttajalle tarkoitettujen mikrokontrollerien kehitysalustojen ja säätösimulaatio-ohjelmistojen monimuotoisuuden ja saatavuuden ansiosta itsenäiset järjestelmäkehittäjät voivat ottaa mielivaltaisen kehitysprojektin automaation säätötekniikkaan liittyen. Tämän mahdollisuuden ansiosta Arduino-pohjaisen robotin säätöjärjestelmä otettiin kehitettäväksi, jotta saataisiin luotua häiriösietoinen säädin, joka kumoaisi käänteisen heilurin vaikutuksen robotissa.

Tässä työssä MinSeg-robotille olemassa oleva säätösuunnitelma tutkittiin huolellisesti ja tehtiin alusta alkaen uudelleen. Aluksi tarvittavat liikeyhtälöt robotin renkaille, rungolle ja moottorille suunniteltiin, linearisoitiin ja yhdistettiin siirtofunktioksi. Sitten saatu mallihypoteesi toteutettiin käytännössä PID-säätimellä, joka stabiloi kyseessä olevan siirtofunktion. Toimivaa säädintä paranneltiin häiriömallin lisäämisellä ja säädön onnistumista tarkkailtiin kenttäkokeilla. Lopuksi PID-säädintä kehitettiin robustimmaksi häiriömallin avulla ensin simulaatioissa ja sitten käytännön kokeissa.

Ratkaisu oli jakaa aihe osiin, joissa aloitettiin säätötekniikan perusteista ja päädyttiin kyseisen tasapainorobotin ohjaukseen. Jokainen matemaattinen säätökomponentti ensin tutkittiin, tehtiin siitä testiversio ja lopuksi kokeiltiin toimivaa ratkaisua käytännössä. Huolellisen ja tarkan kenttätyön ansiosta robotille tallennettiin täsmällisemmät säätöparametrit, minkä ansiosta robotin häiriösietokyky parani.

Avainsanat MinSeg, tasapainottava, robotti, säätö

Preface

I would like to thank Professor Themistoklis Charalambous for the opportunity and the invaluable feedback during the Thesis process.

Espoo, 23.5.2018

Toni K. Silfver

Contents

Symbols and Abbreviations

Symbols

b_f	Friction Coefficient
b_m	Motor Viscous Coefficient
d	Disturbance Input Signal
e	Back Electromotive Force (EMF)
$\mathbf{F}$	The System Vector Sum of All Forces Applied to Each Body [N]
F_t	Tractive Force [N]
F_x	Horizontal Tension Component Between the Wheel and the Body [N]
F_y	Vertical Tension Component Between the Wheel and the Body [N]
g	Gravity [m/s^2]
I_b	Body Moment of Inertia [kgm^2]
i_m	Motor Current [A]
I_w	Wheel Moment of Inertia [kgm^2]
J	Cost Function
k	Time Index for Discrete-Time Systems [s]
K_e	Electric Constant
K_t	Motor Torque Constant
l_b	Distance Between the Body's and the Wheel's Center of Masses [cm]
L_m	Motor Electrical Inductance [H]
l_w	Radius of the Wheel [cm]
m_b	Mass of the Body [kg]
$m_b g$	Gravity of the Body [kgm/s^2]
m_w	Mass of the Wheel [kg]
$m_w g$	Gravity of the Wheel [kgm/s^2]
N	Reaction of the Plane [N]
R_m	Motor Electrical Resistance [Ohm]
t	Time Index for Continuous-Time Systems [s]
T_f	Friction Torque [Nm]
T_m	Motor Torque [Nm]
u	Input Signal
v_m	Motor Input Voltage [V]
x_b	Horizontal Position of the Center of Mass of the Body [cm]
x_w	Horizontal Position of the Center of Mass of the Wheel [cm]
y	Output Signal
y_b	Vertical Position of the Center of Mass of the Body [cm]
y_w	Vertical Position of the Center of Mass of the Wheel [cm]
Δ	Sampling Interval for Discrete-Time Systems [s]
θ_b	Angular Displacement of the Body [rad]
θ_m	Motor Shaft Angle [rad]
θ_w	Angular Displacement of the Wheel [rad]
κ	Alternative Notation for Gain for Continuous-Time Systems [s]

Abbreviations

COM	Communication Port
DC	Direct Current
EMF	Back Electromotive Force
EOM	Equations of Motion
LHP	Left Half-Plane
^{lin}	Linearized
LQR	Linear-Quadratic Regulator
MATLAB	Matrix Laboratory
MIMO	Multi Input Multi Output
MinSeg	Miniature Two-Wheeled Self-Balancing Robot
PD	Proportional-Derivative
PI	Proportional-Integral
PID	Proportional-Integral-Derivative
RHP	Right Half-Plane
Simulink	Simulating Programming Environment
SISO	Single Input Single Output
SRL	Symmetric Root Locus
TF	Transfer Function
USB	Universal Serial Bus
ZOH	Zero-Order Hold

1 Introduction

1.1 Background

The traditional way of arranging the wheels of a transporting device may be seen as a set of two wheels in series or four wheels set in a rectangular disposition from one another. The possibility of a two-wheel human transportation setup where the wheels locate side-by-side has not been invented until the late 90's / early 2000's by the inventor Dean Kamen [9]. With today's computer calculating power and the advancements of control engineering and electronics, one can travel the conventional pedestrian routes by the sheer use of ones body weight distribution. Meaning when the pilot leans forward, the Segway goes forward and when one leans back, it stops or starts to reverse. The problem of balancing an inverted pendulum is similar to the balancing scooter and thus the developed robot control system. The robot is called MinSeg and it is a good development platform for a balancing controller.

Figure 1: The MinSeg robot [49, Section 1].

MinSeg is based on the Arduino platform and Lego Mindstorms. When combined together, it becomes a mini-Segway, which is exactly what the MinSeg balancing robot is. The control methods in this Thesis, explained both mathematically and from a practical point of view, are applicable to any balancing robots and thus could be utilized as the background knowledge for an independent development assignment alike.

1.2 Scope and Objectives

Essentials from the mathematical perspectives to the programming of the Arduino microcontroller and from the simulated model results to the commanding of the real world object MinSeg are covered in this work. Thus the produce was to control the robot so that it would balance itself even when it was subjected to a poke. The simulating objective was to use Simulink for prototyping purposes before finally trying the theories in practice by making a field-test for the robot to see how it behaves in real life. And if possible, make the robot controls more robust by improving the initial PID controller thorough various methods. With poor modelling becomes poor control and with poorly modelled controlled systems it can be impossible to find the real culprit of an event that causes the system to fail. Those kinds of events and many others are not modelled in this Thesis. For the MinSeg, the disturbance effects will be assumed as horizontal pokes by human hand to the center of mass of the robot body and to achieve an easily controllable process, the controller will be designed to be robust enough for such a disturbed environment. There was no need for the Thesis worker to experimentally determine the system characteristics that would then have given the required mathematical framework. The mathematical framework was implemented using an already proven concept as a reference. To derive the Equations of Motion for the full-sized Segway, the beam density, it's deflection at length x along the beam and beam area moment of inertia, would have to be taken into account because there is considerable mass and length in the beam of the Segway [17, Section 2.1]. With the MinSeg, the beam (body) is assumed weightless because it's very light (about 0,114 kilograms) and significantly shorter than the one on a Segway, as it's about the size of a human hand, thus about 22 centimeters in total from which wheels and motor are about 11 centimeters, so none of the aforementioned beam attributes are taken into account in writing the EOM for the MinSeg. Also in this Thesis, there is only one torque for both wheels, as normally in the human sized Segway, there are two torques, one for each wheel [36]. The controller of a Segway is a more complicated PD Lead-Lag [24, 36] type as for the MinSeg, a PID with Linear-Quadratic Regulator (LQR) type of controller will be used. There is a complete study available concerning the real Segway controller design from the publication by Shui-Chun Lin and Ching-Chih Tsai [36].

1.3 Research Methods

The literature mainly used were the MinSeg laboratory guide Version 1.2.1. from Luleå University of Technology by Varagnolo and Lucchese [49] and the seventh edition of Feedback Control of Dynamic Systems by Franklin, Powell and Emami-Naeini [17, Section 2]. MATLAB R2017b version for Microsoft Windows 10 64bit was used as the simulation platform along with the required installation packages for Arduino with the special MinSeg library and the Arduino IDE 1.8.5, which were installed into the operating system for the uploading of the control codes to succeed into the MinSeg Arduino platform. For making the Figures, TechSmith Snagit Editor 13.0.2 was used for drawing and TechSmith Snagit 13 was used for picture

capturing. For additional reference, wikipedia pages were visited and youtube videos were watched. Both the laboratory guide and the seventh edition were thoroughly examined and by the end of this work, the target of attaining a robust controller for the robot was reached.

1.4 Thesis Structure

This Thesis is divided into five main chapters and a summarizing sixth chapter. Chapter 1 explains the background, the Thesis objectives, current research and the motivation for this work. Chapter 2 covers the mathematics behind the balancing robot control theory, which is the basis of designing the PID controller and the most important part of this work. The aim of the mathematical framework is to make the robot's controller as robust as possible with the PID derived from the mathematical theory in order to counter the adverse effects of a small disturbance signal. Chapter 3 shows how to turn the theories into practice by developing the PID controller and implementing it to the robot and observing the results in a field-test. Chapter 4 improves the PID by designing and adding a State-Space controller and a state observer to it. After this the PID is again put thorough a field-test. Chapter 5 teaches how to re-design the PID directly on discrete-time domain so that the design process would take less time in making any future PID improvements as preliminary calculations from the continuous-time domain are not necessary anymore. Finally chapter 6 gives the conclusions attained from this work.

2 The Mathematical Framework of the MinSeg Robot

The purpose of the mathematical model for the MinSeg robot is to help in developing a simulator in which the robot can be tested for prototyping purposes [49, Section 3.1]. To be able to achieve this, the robot must be described mathematically, which is the first task of the control design process. After all, without the profound theories of control, the robot wouldn't even stand up straigh as this would be the same thing as trying to balance an inverted pendulum. So it is necessary to describe the robot mathematically in the beginning of the Thesis. The mathematical description of the MinSeg is just a ring and a rod with a point mass, which is adequate for making the mathematics work, as it resembles a pendulum. The robot has to be able to follow a certain reference signal set by the user (a person) even if there were disturbances, so that the user would have a control over the robot. When a controllable object follows a reference signal, like an eye following a flying ball or a thermostat keeping a room temperature to a set level, then it would be an example of a feedback controlled process. [17, Section 2] A feedforward controller, on the other hand, can predict disturbances, which can be handled as predetermined events. This becomes especially handy in countering the effects of disturbances and when combined with a feedback controller, will make the balancing robot process more robust and well controlled relative to the expected operating environment compared to operating solely with either one. A disturbance can also be anything not predetermined in the mathematical model of the robot, e.g. an unexpected electric motor failure or something catastrophic, like incineration. If the modelled system does not behave as expected in real life, it is possible that the controller is not robust enough. Eventually the process control of the MinSeg will be done by a feedback-feedforward controller, which uses the best abilities of both controller types [48, Section 1.3].

2.1 The Equations of Motion

2.1.1 Introduction

As the robot is a physical object, the Equations of Motion (EOM) must be created by using the aforementioned mathematical description as its guide. The EOM constitute the parameters that are the commands to the robot for one to be able to control it where to go. The parameters are given to the robot via a USB cable provided in the MinSeg package or by its bluetooth module that resides in the robot's blue Arduino board. Both USB and Bluetooth connections are established on a PC that runs a recent MATLAB program. As the name MATLAB (MATrix LABoratory) suggests, this program uses matrices to perform its calculations and the robot parameters are stored in them. The matrices however are constructed in one or more scripts that are connected to a simulator (Simulink), which in turn is a software extension of MATLAB. In Simulink, the results of the commands to the robot are simulated and they operate by user friendly functional block diagrams, which represent the different functions of the corresponding simulated system. To derive the EOM for the MinSeg

(or another similar unstable dynamical system), these five steps [17, Section 2.1] are involved:

1. Assign suitable variables e.g. x or θ that are necessary to describe the object sufficiently.
2. Draw a free-body diagram [19] to each body component and show their forces and force reference directions including the center of mass acceleration with respect to an inertial reference frame.
3. As for a rigid body like the MinSeg, the Newton's laws of motions are enough to describe the physics, so use them to derive the Equations of Motion.
4. Then eliminate the internal forces by combining all the derived movement equations.
5. And finally check that there are as many Equations of Motion that there are unknown variables.

The initial observation of the MinSeg is that it is a mechanical system. A mechanical system is a system that thorough power management accomplishes a task that involves movement and forces [26]. The forces derived from the mechanical system give us the mathematical description of the robot that is embedded in the Equations of Motion. These forces that describe the laws of nature useful for our purposes, were invented by Sir Isaac Newton already in the seventeenth century [29]. To a mechanical system, the Newton's second law is the key:

$$\mathbf{F} = m\mathbf{a}, \tag{1}$$

where $\mathbf{a}$ is the vector acceleration with respect to the stars i.e. with respect to an inertial reference frame. By selecting the respective coordinates this way, we only need to derive the robot position twice to get its acceleration. Vector acceleration simply means that the acceleration has a magnitude and an orientation. A boldfaced text here indicates a vector quantity and that the vector consists of other vectors, vector functions or matrices. To derive the EOM, it should be clear what the mathematical description is as an illustration. In case of the MinSeg, this would be a rod accompanied by a point (point mass) at its upper end and a ring at its lower end. A point mass is a point where all of the mass is travelling at the same radius in a circle that represents the body, whereas the ring represents the wheel(s) of the robot. The forces acting on the body of the robot (position, velocity, acceleration, rotation and disturbance) are sketched out using a free-body diagram [18] presented in Figure 2. The derivation of the EOM is convenient by referring to the free-body diagram of the force directions and other issues concerning the generalized representation of the robot system. The equation (1) can be modified to look like this:

$$\mathbf{M} = \mathbf{I}\alpha, \tag{2}$$

where $\mathbf{M}$ [Nm] is the sum of external moments affecting the body, $\mathbf{I}$ is the body's mass moment of inertia and α [rad/s] is the angular acceleration of the rotating

body. This formula (2) is used for rotational systems. The moment of inertia is a rotational inertia, which is handy in torque calculations, as torque is a force that causes something to start rotating. The **I** tells how difficult it would be to angularly accelerate an object and the angular acceleration is the rate at which an object would start to accelerate from rest. To angularly accelerate something, a force would be required that is tangential to its circle orbit at the very edge of which the point mass is rotating. [17, Section 2.1] [27] [49, Section 1]

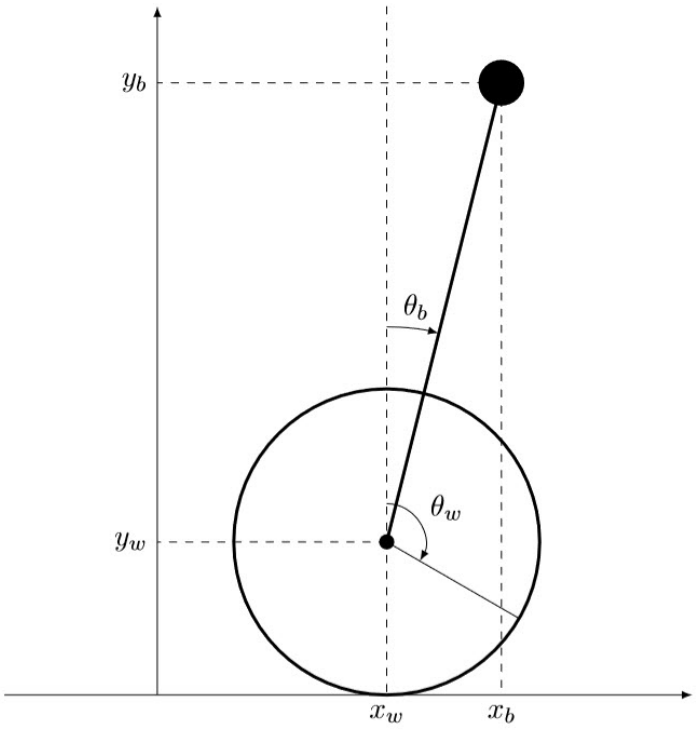

Figure 2: The MinSeg schematic representation [49, Section 1].

In order to reach the correct resulting EOM, the robot system should be expressed in this way:

$$\begin{cases} \ddot{x}_w = f_{x_w}(x_w, \dot{x}_w, \theta_b, \dot{\theta}_b, \ddot{\theta}_b, v_m) \\ \ddot{\theta}_b = f_{\theta_b}(x_w, \dot{x}_w, \ddot{x}_w, \theta_b, \dot{\theta}_b, v_m) \end{cases} \tag{3}$$

In the equations of (3), the appropriate (suitable) functions, for which to begin with the derivation of the Equations of Motion, are for wheel position f_{x_w} and body angle f_{θ_b}, which besides themselves, are composed of other functions written here inside the parenthesis. The two functions are saved in two variables and correspond to two equations respectfully as seen in (3). The two variables are acceleration of the wheel $\ddot{x}_w$ and acceleration of the body angle $\ddot{\theta}_b$. In the later chapters, the wheel angle and body position functions will be utilized with the wheel position and body angle functions in order to have the final EOM derived. The surface on which the MinSeg moves is assumed to be a flat, horizontal environment where the vertical acceleration of the wheel $\dot{y}_w$ is always zero. The single dot on top of the vertical velocity of the wheel term $\dot{y}_w$ indicates the first derivative of the wheel position,

which is the wheel velocity and two dots would indicate the second derivative, which is the wheel's vertical acceleration. In addition to the wheel surface assumptions, no slip of the wheel or turning of the robot, while in operation, by any means of the user, are preferred. Thus the wheel position is assumed to be linear and equal to the product of the wheel radius and the wheel angle

$$x_w = l_w \theta_w.$$

There are no aerodynamics involved in the calculations and the controller is developed to control the input voltage. When voltage is applied via the batteries or the USB cable to the robot's motor voltage input port, the motor shaft turns, which creates torque that is applied to the wheel. MinSeg needs at least two wheels to operate as expected, i.e. to balance its body by moving back an forth on a grippy, level surface. But for the purpose of calculation simplicity, the two wheels are represented as one wheel as seen in Figure 2 schematic. The wheel dynamics are derived independently, while keeping the motor torque T_m inside the equations for future use. The body dynamics are also derived independently from wheel dynamics while keeping the motor torque T_m inside the equations. Then the wheel and body dynamics are combined and motor input voltage v_m is replaced to motor torque T_m in the EOM. The motor torque affects body forces, which apply to the center of mass (or body) dynamics. The motor torque is equal to torque dynamics, which is the key element in designing the feedback controller for the robot because the robot is controlled by motor input voltage and thus the torque dynamics. The derivation of the motor dynamics is necessary, because the motor voltage affects wheel torque. When torque dynamics T_m are derived as a function of v_m and inserted to the combined dynamics, the result is the motor dynamics, which in turn is the result of developing an input voltage controlled system. [49, Section 3.1]

2.1.2 Wheel Dynamics

Now, after the introduction is completed, the first task is to model the dynamics of the wheel as the wheel dynamics affect body forces [49, Section 3.1.1]. Hence the forces applying to the wheel are now of interest.

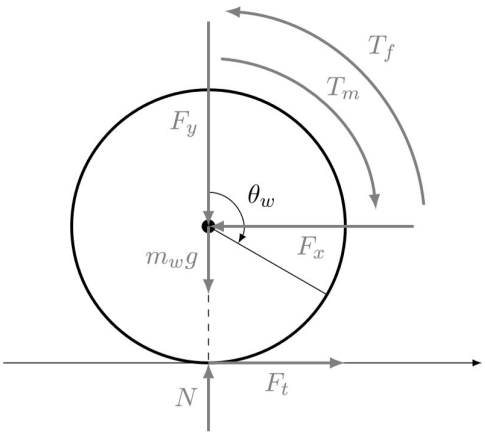

Figure 3: Forces applying to the wheel [49, Section 3.1.1].

As seen in Figure 3, all essential forces applying to the wheel are the wheel-body vertical tension F_y, wheel-body horizontal tension F_x, wheel gravity $m_w g$, reaction of the plane N, tractive force (motor-wheel pulling capability) F_t, friction torque T_f, motor torque T_m and wheel angular displacement θ_w. In the case of the wheel, the suitable functions for this first step of modelling the EOM are the wheel position f_{x_w} and wheel angle f_{θ_w}. To derive the wheel dynamics for x_w, y_w and θ_w, the external forces

$$\mathbf{F} = \{T_m, T_f, F_x, F_y, F_t, m_b g\}$$

come into play and the dynamics of the wheel system can be expressed like the following (4). [49, Section 3.1.1]

$$\begin{cases} \ddot{x}_w = f_{x_w}(x_w, \dot{x}_w, \theta_w, \dot{\theta}_w, \mathbf{F}) \\ \ddot{\theta}_w = f_{\theta_w}(x_w, \dot{x}_w, \theta_w, \dot{\theta}_w, \mathbf{F}) \end{cases} \tag{4}$$

The wheel equations of (4) contain the same elements in both suitable functions inside the parenthesis as both wheel position and wheel angle are affected by the same forces. The vector quantity $\mathbf{F}$ indicates the external forces and has changed the elements compared to formula (3). The reason why the acceleration of the wheel $\ddot{x}_w$ has different elements here, is that the wheel formula (4) is only a part of the system formula (3) and is affected by only the forces applying to the wheel and not also the body. The external forces in (4) embedded in the vector quantity $\mathbf{F}$ are almost the same as all the forces applying to the wheel of Figure 3. Exceptions to this are the body gravity $m_b g$, which is an external force and not accountable as the internal force of the wheel, and reaction of the plane N that is caused by the interaction of the wheel to the wheel surface, which has the same magnitude as $m_w g$ but that is directed perpendicularly towards the wheel from the surface. All in all there are three types of forces affecting the wheel: vertical (agreed to be null), horizontal and angular. The vertical velocity of the MinSeg is set to be zero, thus velocity and acceleration (and the wheel position at the ground level) of the wheel are zero to the same direction. By utilizing the Newton's second law of motion formula (1) and looking at Figure 3 of the wheel, we can see that the forces of the wheel equal to zero towards the vertical direction in equation (5). [49, Section 3.1.1]

$$N - m_w g - F_y = m_w \ddot{y}_w = 0 \tag{5}$$

The horizontal movement forces are easily derived the same way seen in equation (6), but now these forces are included in the mathematical model as they are not equal to zero [49, Section 3.1.1].

$$F_t - F_x = m_w \ddot{x}_w \tag{6}$$

The angular movement forces of the wheel use the formula (2) for rotational systems [49, Section 3.1.1].

$$T_m - T_f - l_w F_t = I_w \ddot{\theta}_w \tag{7}$$

In formula (7) the I_w represents the rotational inertia $\mathbf{I}$ from Newton's rotational system formula (2), $\ddot{\theta}_w$ the angle α and the rest of the equation the forces adding

to the equation (motor torque T_m) and forces deducting from the equation (friction torque T_f, wheel radius l_w and tractive force F_t) [49, Section 3.1.1].

2.1.3 Body Dynamics

The second task is to model the dynamical equations for the body. As seen in Figure 4, the forces and their direction acting on the body are $T_m, T_f, F_x, F_y, m_b g$ and θ_b from which external ones are

$$\mathbf{F} = \{T_m, T_f, F_x, F_y, m_b g\}.$$

The only difference here to the external forces of the wheel motion is the absence of the tractive force F_t. This happens because the motor only pulls the wheel forward, not the body, as the motor shaft is turning the wheel [49, Section 3.1.2].

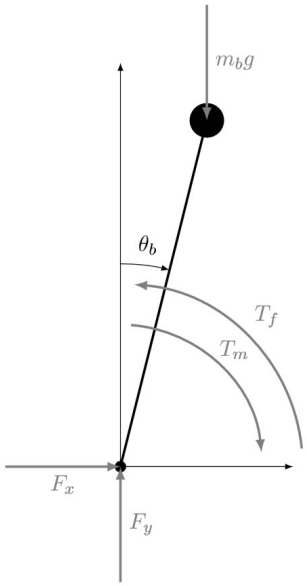

Figure 4: Forces applying to the body [49, Section 3.1.2].

When the motor torque T_m is in effect, θ_b is affected to the negative direction and when torque friction T_f is at work then θ_b is affected to the positive direction [49, Section 3.1.2]. The positive and negative directions of forces can be freely chosen, which ever fits the engineering better.

$$\begin{cases} \ddot{x}_b = f_{x_b}(x_b, \dot{x}_b, \theta_b, \dot{\theta}_b, \mathbf{F}) \\ \ddot{\theta}_b = f_{\theta_b}(x_b, \dot{x}_b, \theta_b, \dot{\theta}_b, \mathbf{F}) \end{cases} \tag{8}$$

The focus now is directed towards the suitable functions and variables for the body dynamics keeping in mind that the goal is to reach a similar representation of the system as in equation (3). In fact, the representation of (8) is the same as (4) except the functions for the wheel ($_w$) are now expressed for the body ($_b$). When examining Figure 4, one can notice its similarity to a unit circle [45]. Figure 4 can be imagined to have polar coordinates [32] when the moving point mass would move in a circle

radius. The parametric equations for a circle are given by the moving point (point mass or center of mass on the body) and the horizontal component

$$x = rcos\theta,$$

which describes the moving point position in horizontal direction and vertical component

$$y = rsin\theta,$$

which describes the moving point position in vertical direction. The θ term is the angle of the moving point in a circle and r is the circle radius [31]

$$\sqrt{x^2 + y^2}.$$

With these equations as a starting point, it's easy to derive the equations for the body. Again, three separate equations are needed to derive the motion equations for the suitable functions x_b and θ_b. One for vertical, one for horizontal and one for angular movement, just like in the wheel. In the vertical direction, the forces for the movement of the body are affected by gravity as the vertical wheel and body tension component (F_y) is not perpendicular to it, so the body gravity force is deducted from the equation. The mass element is the mass of the body and the vertical acceleration is presented as $\ddot{y}_b$ in accordance to Newton's second law of motion, which then derives into equation (9) [49, Section 3.1.2].

$$F_y - m_b g = m_b \ddot{y}_b \tag{9}$$

For the horizontal dynamics gravity has no effect, as they are perpendicular to each other and the system point of rotation is the center of the mass, which is in the body. So $m_b g$ is absent in the formula (10) [49, Section 3.1.2].

$$F_x = m_b \ddot{x}_b \tag{10}$$

For the rotational inertia (moment of inertia), the gravity has no effect, because it does not affect torque in any way, as MinSeg motor torque is generated by motor input voltage and not by gravity. The body angle term θ_b has no gravity effect because of zero torque effects. Torque is exposed to mechanical frictions and is proportional to the rotational velocity of the motor. Angular dynamics are affected only by the motor torque T_m and torque friction T_f [49, Section 3.1.2]. The body length variable l_b is needed now instead of l_w, just like in the wheel's case, and also the torque terms T_m and T_f have to be included in the rotational body equation as they are apparent in Figure 4. From Newton's second law of motion in equation (2), $\mathbf{I}\alpha$ is converted to $I_b\ddot{\theta}_b$. We already know the positive and negative directions of movements, which have been set to our own preference being $-T_m$ and $+T_f$. The next thing that is needed is to decide the derivation angle of the right triangle to which all the sine and cosine clauses would be based upon [19]. It turned out by trial and error that this angle should be set on the center of mass (point mass) as it should be the rotational axis of the MinSeg system. This makes perfect sense

when imagining an inverted pendulum, which the MinSeg resembles. The point of rotation on an inverted pendulum is set to its base, which would be the attachment point of the pendulum itself. By keeping this in mind, it is easy to imagine the MinSeg balancing itself. When the robot is balancing, the wheels move back and forth while the upmost point of the robot stays still, just like an inverted pendulum at the vertically upright position. In Figure 5, the agreed positive and negative axis are shown that contribute to the signs of the F_x and F_y terms in the body rotational equation. The right triangle shows the position of the point mass (the big black dot) and the set angle from which the equation is derived using the SOHCAHTOA [39] method. The y and x terms in Figure 5 represent the vertical and horizontal axis with respect to the surface and the term l_b points towards the center of the wheel from the point mass perspective. With SOHCAHTOA, the $l_b \sin(\theta_b)$ along the y-axis is derived from

$$\sin(\theta_b) = \frac{y}{l_b}$$

and the $l_b \cos(\theta_b)$ along the x-axis is derived from

$$\cos(\theta_b) = \frac{x}{l_b}.$$

By multiplying these terms according to the set positive and negative directions along x- and y-axis respectively, the outcome is $+F_y l_b \sin(\theta_b) - F_x l_b \cos(\theta_b)$ and thus we get the following expression in equation (11).

$$T_f - T_m + F_y l_b \sin(\theta_b) - F_x l_b \cos(\theta_b) = I_b \ddot{\theta}_b \tag{11}$$

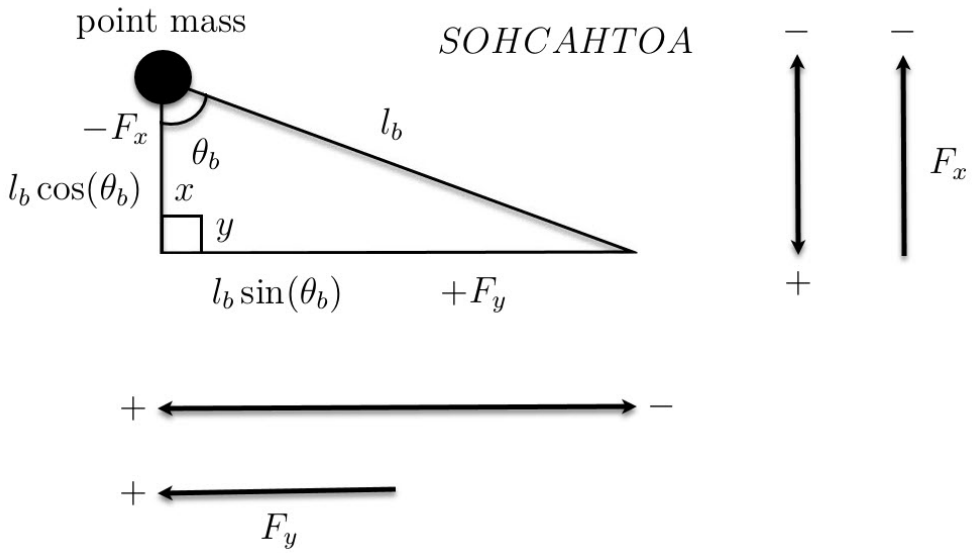

Figure 5: Deriving the Body Equations [19].

The $-F_x$ component is perpendicular to the point mass and is the one causing the rotation of the robot body. The $-F_y$ component is parallel to the point mass (or tangential to the circle) and is the other accompanying part in the total body force

effect. The angle θ is always the angle between the combined force vector of F_x, F_y and the body length variable l_b. When using multiple forces, it is important to check that the signs are correct. Point mass represents the point where the force is applied. It is assumed to be applied to the very edge of the imaginary circle made by a rotating pendulum, so that the whole radius (l_b) is used. Otherwise other radiuses should be considered, which would be the case if the MinSeg was designed longer or shorter. With bigger radius or bigger body mass, the MinSeg would be harder to angularly accelerate, because the rotational inertia would also grow bigger. The angular acceleration $\ddot{\theta}_b$ is the rate at which the MinSeg robot's body would start to accelerate if it started from rest and tell the rate at which the angular velocity is changing [16]. We can get closer to equation (3) with equations (4) and (8) thorough system equations in (12). The aim now is to eliminate other variables than the wheel position x_w and body angle θ_b, because they are the suitable variables in equation (3) and write them as functions of x_w, θ_b, T_m and T_f.

$$\begin{cases} \ddot{x}_w = f_{x_w}(x_w, \dot{x}_w, \theta_b, \dot{\theta}_b, \ddot{\theta}_b, T_m, T_f) \\ \ddot{\theta}_b = f_{\theta_b}(x_w, \dot{x}_w, \ddot{x}_w, \theta_b, \dot{\theta}_b, T_m, T_f) \end{cases} \tag{12}$$

Internal tension forces F_x, F_y are supposed to disappear from the EOM according to the fourth step of the five steps of deriving the EOM, so these forces will need to be expressed in an alternative way. This is the case for the tractive force F_t also. To this outcome, F_x, F_y in formula (11) can be eliminated with the help of equations (9) and (10) and thus advance to a new equation of (13) [49, Section 3.1.2].

$$T_f - T_m + m_b l_b g \sin(\theta_b) + m_b l_b \ddot{y}_b \sin(\theta_b) - m_b l_b \ddot{x}_b \cos(\theta_b) = I_b \ddot{\theta}_b \tag{13}$$

Compared to equation (11), in the new equation (13), $F_y l_b \sin(\theta_b)$ was changed to $m_b l_b g \sin(\theta_b) + m_b l_b \ddot{y}_b \sin(\theta_b)$ and $F_x l_b \cos(\theta_b)$ was changed to $m_b l_b \ddot{x}_b \cos(\theta_b)$, while the rest of the equation stayed the same. Now F_x and F_y are eliminated with the exploitation of (9) and (10). But (13) would be a bad expression if there were x_b and y_b terms, because we don't need to know body vertical and horizontal positions at this time, so there becomes a need to simplify the expression $\ddot{y}_b \sin(\theta_b) - \ddot{x}_b \cos(\theta_b)$ in equation (13). For this we can use the derivative rules [10] for the product rule and chain rule parts. The product rule [10] is

$$\frac{d}{dx} fg = fg' + f'g$$

and the chain rule [10] is

$$\frac{d}{dx} f(g(x)) = f'(g(x))g'(x).$$

To achieve the simplified result for the expression $\ddot{y}_b \sin(\theta_b) - \ddot{x}_b \cos(\theta_b)$, one can take advantage of the equations in (14), which are results of the affecting MinSeg force

dynamics.

$$\begin{cases} x_b = x_w + l_b \sin(\theta_b) \\ \dot{x}_b = \dot{x}_w + \dot{\theta}_b l_b \cos(\theta_b) \\ \ddot{x}_b = \ddot{x}_w + \ddot{\theta}_b l_b \cos(\theta_b) - \dot{\theta}_b^2 l_b \sin(\theta_b) \\ y_b = y_w + l_b \cos(\theta_b) \\ \dot{y}_b = \dot{y}_w - \dot{\theta}_b l_b \sin(\theta_b) = -\dot{\theta}_b l_b \sin(\theta_b) \\ \ddot{y}_b = -\ddot{\theta}_b l_b \sin(\theta_b) - \dot{\theta}_b^2 l_b \cos(\theta_b) \end{cases} \tag{14}$$

Considering the horizontal body movements, we know that the derivative of x_b is $\dot{x}_b$ and the derivative of x_w is $\dot{x}_w$. For the first equation in (14)

$$x_b = x_w + l_b \sin(\theta_b)$$

and by using the chain rule, in $\frac{d}{dx}(l_b \sin(\theta_b))$ the θ_b is selected as $g(x)$ and $\sin(\theta_b)$ as $f(g(x))$. Thus the derivation of $l_b \frac{d}{dx} \sin(\theta_b)$ results in $l_b(\cos(\theta_b)\dot{\theta}_b)$, which in turn is $\dot{\theta}_b l_b \cos(\theta_b)$. The resulting equation is the second equation

$$\dot{x}_b = \dot{x}_w + \dot{\theta}_b l_b \cos(\theta_b).$$

The derivative of $\dot{x}_b$ is $\ddot{x}_b$ and the derivative of $\dot{x}_w$ is $\ddot{x}_w$. For the second equation and by using the product and chain rules, in $\frac{d}{dx}(\dot{\theta}_b l_b \cos(\theta_b))$ the $\cos(\theta_b)$ is selected as $(g(x))g'(x)$ and $\dot{\theta}_b l_b$ as $f'(g(x))g'(x)$. Thus the derivation results in $\dot{\theta}_b l_b(-\sin(\theta_b))\dot{\theta}_b + \ddot{\theta}_b l_b \cos(\theta_b)$, which in turn is $\ddot{\theta}_b l_b \cos(\theta_b) - \dot{\theta}_b^2 l_b \sin(\theta_b)$. The resulting equation is then the third equation

$$\ddot{x}_b = \ddot{x}_w + \ddot{\theta}_b l_b \cos(\theta_b) - \dot{\theta}_b^2 l_b \sin(\theta_b).$$

Considering the vertical body movements, we know that the derivative of y_b is $\dot{y}_b$ and the derivative of y_w is $\dot{y}_w$. For the fourth equation

$$y_b = y_w + l_b \cos(\theta_b)$$

and by using the chain rule, in $\frac{d}{dx}(l_b \cos(\theta_b))$ the θ_b is selected as $g(x)$ and $\cos(\theta_b)$ as $f(g(x))$. Thus the derivation of $l_b \frac{d}{dx} \cos(\theta_b)$ results in $l_b(-\sin(\theta_b)\dot{\theta}_b)$, which in turn is $-\dot{\theta}_b l_b \sin(\theta_b)$. The resulting equation is the fifth equation

$$\dot{y}_b = -\dot{\theta}_b l_b \sin(\theta_b)$$

from which the $\dot{y}_w$ term is zero, as the vertical velocity of the wheel is agreed to be null. The derivative of $\dot{y}_b$ is $\ddot{y}_b$. For the fifth equation and by using the product and chain rules, in $\frac{d}{dx}(-\dot{\theta}_b l_b \sin(\theta_b))$ the $\sin(\theta_b)$ is selected as $(g(x))g'(x)$ and $-\dot{\theta}_b l_b$ as $f'(g(x))g'(x)$. Thus the derivation results in $-\dot{\theta}_b l_b(\cos(\theta_b))\dot{\theta}_b - \ddot{\theta}_b l_b \sin(\theta_b)$, which in turn is $-\ddot{\theta}_b l_b \sin(\theta_b) - \dot{\theta}_b^2 l_b \cos(\theta_b)$. The resulting equation is then the sixth equation

$$\ddot{y}_b = -\ddot{\theta}_b l_b \sin(\theta_b) - \dot{\theta}_b^2 l_b \cos(\theta_b).$$

Variables $\ddot{y}_b$ and $\ddot{x}_b$ are relative to $\ddot{\theta}_b$ and $\ddot{x}_w$ as can be seen from (14). This relation helps to simplify equation (13). [49, Section 3.1.2]

$$\begin{aligned}\ddot{y}_b \sin(\theta_b) - \ddot{x}_b \cos(\theta_b) &= \Big(- \ddot{\theta}_b l_b \sin(\theta_b) - \dot{\theta}_b^2 l_b \cos(\theta_b) \Big) \sin(\theta_b) \\ &\quad - \Big(\ddot{x}_w + \ddot{\theta}_b l_b \cos(\theta_b) - \dot{\theta}_b^2 l_b \sin(\theta_b) \Big) \cos(\theta_b) = -\ddot{\theta}_b l_b \sin^2(\theta_b) \\ &\quad - \dot{\theta}_b^2 l_b \cos(\theta_b) \sin(\theta_b) - \ddot{x}_w \cos(\theta_b) - \ddot{\theta}_b l_b \cos^2(\theta_b) + \dot{\theta}_b^2 l_b \sin(\theta_b) \cos(\theta_b) \\ &\quad = -\ddot{\theta}_b l_b - \ddot{x}_w \cos(\theta_b)\end{aligned} \tag{15}$$

In (15) with respect to (14), the $\ddot{y}_b$ term is changed to $-\ddot{\theta}_b l_b \sin(\theta_b) - \dot{\theta}_b^2 l_b \cos(\theta_b)$ according to the third equation of (14) and $\ddot{x}_b$ is changed to

$$\ddot{x}_b = \ddot{x}_w + \ddot{\theta}_b l_b \cos(\theta_b) - \dot{\theta}_b^2 l_b \sin(\theta_b)$$

according to the sixth equation of (14). The equations in (15) simplify $\ddot{y} \sin(\theta_b) - \ddot{x}_b \cos(\theta_b)$ into $-\ddot{\theta}_b l_b - \ddot{x}_w \cos(\theta_b)$. When equation (15) is plugged into (13) we get (16).

$$T_f - T_m + m_b l_b g \sin(\theta_b) - m_b l_b^2 \ddot{\theta}_b - m_b l_b \ddot{x}_w \cos(\theta_b) = I_b \ddot{\theta}_b \tag{16}$$

Equation (16) can be rearranged to get (17).

$$m_b l_b g \sin(\theta_b) - m_b l_b \ddot{x}_w \cos(\theta_b) - T_m + T_f = (I_b + m_b l_b^2) \ddot{\theta}_b \tag{17}$$

We can utilize the well known angular velocity formula

$$\omega = \frac{v}{r}$$

to our advantage, where ω is the angular velocity $\ddot{\theta}_w$, v is the linear velocity $\ddot{x}_w$ and r is the circular path radius l_w. [3]

$$\ddot{\theta}_w = \frac{\ddot{x}_w}{l_w} \tag{18}$$

Equation (18) is very useful for wheel angular movement and by plugging (6) into (7) and back into (18) results in (19).

$$T_m - T_f - l_w F_x - l_w m_w \ddot{x}_w = \frac{I_w}{l_w} \ddot{x}_w \tag{19}$$

To eliminate F_x in (19), one needs to combine (10) and (14) and get (20).

$$F_x = m_b \ddot{x}_w + m_b l_b \ddot{\theta}_b \cos(\theta_b) - m_b l_b \dot{\theta}_b^2 \sin(\theta_b) \tag{20}$$

When (20) is plugged into (19), we get:

$$T_m - T_f - l_w \Big(m_b \ddot{x}_w + m_b l_b \ddot{\theta}_b \cos(\theta_b) - m_b l_b \dot{\theta}_b^2 \sin(\theta_b) \Big) - l_w m_w \ddot{x}_w = \frac{I_w}{l_w} \ddot{x}_w \tag{21}$$

And when rearranged, we get:

$$-m_b l_b l_w \ddot{\theta}_b \cos(\theta_b) + m_b l_b l_w \dot{\theta}_b^2 \sin(\theta_b) - T_f + T_m = \left(\frac{I_w}{l_w} + l_w m_b + l_w m_w\right)\ddot{x}_w \quad (22)$$

Equation (22) shows the current stage of the derivation of the EOM. But we still have to derive the motor dynamics and add them to (22) before we are finished for the preliminary mathematics part of the MinSeg.

2.1.4 Motor Dynamics

The MinSeg motor is assumed to be rigidly attached to body, which helps to avoid some involved equations about the flexible dynamics of a joint beam [17, Section 2.1]. Although in real life the body of the motor (i.e. the 11 centimeter long leg of the Lego Mindstorms motor) is attached to the body (the Arduino microcontroller board) by male and female Lego parts and is really not very rigid as the two counterparts are not glued together or by other means attached rigidly to each other. The parts disassemble by pulling by hand with minimal force and also might come apart by gravity and impact to the surface if the robot fell down on its face or in the rearward with a full stack of batteries on its back. For the MinSeg, we don't need to make the mathematics too fuzzy, but instead try to keep things as simple and integral as possible.

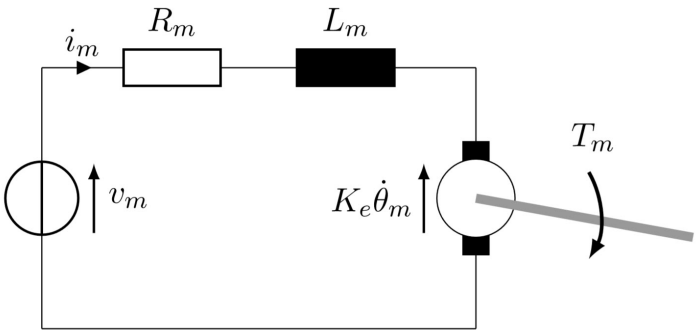

Figure 6: A DC-motor schematic representation [49, Section 3.1.3].

In Figure 6, the motor electrical inductance L_m is assumed to be zero, because when the motor is at rest, there is no inductance in the coil as it is assumed to have discharged itself thorough the closed circuit after the previous motor run. Also the motor angular velocity $\dot{\theta}_m$ indicates the motor angle, which is a function of wheel angle θ_w and body angle θ_b [49, Section 3.1.3].

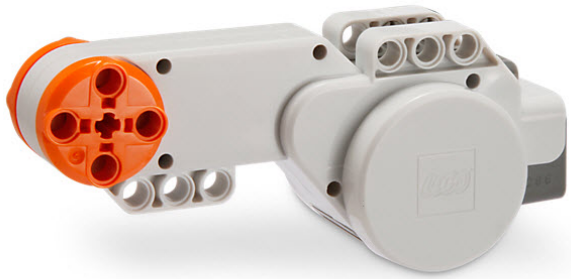

Figure 7: The Lego Mindstorms NXT 2.0 motor (9842), a retired product [44].

The brushed Direct Current servomotor itself, which is displayed in Figure 7, is made out of various parts. The motor is a battery or 5 Volt USB-charge powered Direct Current DC-motor. Common to normal brushed DC-motors, there is a carbon brush, which is in contact with the rotating part of the motor, the commutator, which in turn generates electricity to the rotor's copper windings by the influence of the static stator magnets [17, Section 2.3.2]. Compared to brushless and thus more expensive motors, the brushed motors are usually simpler, cheaper and wear out faster because of the internal friction caused by rubbing of the brushes to the commutator. Friction torque T_f can be expressed in another way by using the friction coefficient b_f and wheel and body angle velocities. With the help of equation (17), the wheel angle velocity θ_w can be expressed as $\frac{\dot{x}_w}{l_w}$ as seen in (23) [49, Section 3.1.3].

$$T_f = b_f\left(\dot{\theta_w} - \dot{\theta_b}\right) = b_f\left(\frac{\dot{x}_w}{l_w} - \dot{\theta_b}\right) \tag{23}$$

As mentioned earlier, the friction coefficient b_f, in (23), is proportional to the motor velocity, thus torque friction T_f can be dropped off in equation (24) [49, Section 3.1.3].

$$\begin{cases} \ddot{x}_w = f_{x_w}(x_w, \dot{x}_w, \theta_b, \dot{\theta}_b, \ddot{\theta}_b, T_m) \\ \ddot{\theta}_b = f_{\theta_b}(x_w, \dot{x}_w, \ddot{x}_w, \theta_b, \dot{\theta}_b, T_m) \end{cases} \tag{24}$$

This is because zero motor velocity derives into zero friction torque ($T_f = 0$). When the MinSeg is at the equilibrium, the motor is at rest and doesn't move or change its motor angle velocity $\dot{\theta}_m$ from zero to non-zero. The target is to derive the EOM in the equilibrium state, which is the steady state. The next task is to express the motor torque T_m as a function of motor voltage v_m in order to arrive at an equal representation of (3). In order for this task to be completed, the relation of the motor torque T_m needs to be expressed with the motor input voltage, motor velocity and body angle velocity (respectfully $v_m, \dot{x}_m$ and $\dot{\theta}_b$). In equation (25), this relation is presented with the suitable function variable f_{T_m}. The motor viscous coefficient b_m is set to be negligible, because it is an internal force component and destined to be eliminated from the EOM, even though in reality for a brushed motor, there really is

internal tension inside the motor. Thus the b_m term doesn't involve in the equations [49, Section 3.1.3].

$$T_m = f_{T_m}\left(v_m, \dot{x}_w, \dot{\theta}_b\right) \tag{25}$$

From the motor schematic in Figure 6, one can derive equation (26) using the guidance of Chapter 2.3.2 from the seventh edition by Franklin, Powell and Emami-Naeini [17, Section 2.3.2].

$$L_m \frac{di_m}{dt} + R_m i_m = R_m i_m = v_m - e \tag{26}$$

In equation (26), the coil term L_m has been dropped off as it is zero and we end up with a simpler equation where the motor electrical resistance R_m is multiplied by motor current i_m according to the Kirchoff's second circuit law [23]. The back electromotive force e (EMF) connects to the motor angular velocity, as seen in (27).

$$e = K_e\left(\dot{\theta_w} - \dot{\theta_b}\right) = K_e\left(\frac{\dot{x}_w}{l_w} - \dot{\theta}_b\right) \tag{27}$$

The given combined term of $R_m i_m$ is equal to the motor voltage deducted by the back-emf element e. By substituting (27) to (26) we have another representation of the motor current in equation (28) [49, Section 3.1.3].

$$i_m = \frac{v_m}{R_m} - \frac{K_e}{R_m}\left(\frac{\dot{x}_w}{l_w} - \dot{\theta}_b\right) \tag{28}$$

Because the motor current i_m is equal to motor torque T_m divided by motor torque constant K_t, the equation (28) becomes (29).

$$T_m = \frac{K_t}{R_m} v_m - \frac{K_e K_t}{R_m}\left(\frac{\dot{x}_w}{l_w} - \dot{\theta}_b\right) \tag{29}$$

Now the EOM is derived into the desired form of (3) and this can be seen completed in the new equations of (30). In the upper equation of (30), the terms $-T_f + T_m$ from (22) have changed to $+\frac{K_t}{R_m} v_m - \left(\frac{K_e K_t}{R_m} + b_f\right)\left(\frac{\dot{x}_w}{l_w} - \dot{\theta}_b\right)$ according to (23) and (29) respectively. And in the lower equation, the terms $-T_m + T_f$ in (17) have changed to $-\frac{K_t}{R_m} v_m + \left(\frac{K_e K_t}{R_m} + b_f\right)\left(\frac{\dot{x}_w}{l_w} - \dot{\theta}_b\right)$ according to (29) and (23) respectively [49, Section 3.1.3].

$$\begin{cases} \left(\frac{I_w}{l_w} + l_w m_b + l_w m_w\right)\ddot{x}_w = \\ -m_b l_b l_w \ddot{\theta}_b \cos(\theta_b) + m_b l_b l_w \dot{\theta}_b^2 \sin(\theta_b) + \frac{K_t}{R_m} v_m - \left(\frac{K_e K_t}{R_m} + b_f\right)\left(\frac{\dot{x}_w}{l_w} - \dot{\theta}_b\right) \\ \\ (I_b + m_b l_b^2)\ddot{\theta}_b = \\ +m_b l_b g \sin(\theta_b) - m_b l_b \ddot{x}_w \cos(\theta_b) - \frac{K_t}{R_m} v_m + \left(\frac{K_e K_t}{R_m} + b_f\right)\left(\frac{\dot{x}_w}{l_w} - \dot{\theta}_b\right) \end{cases} \tag{30}$$

The next thing to do, is to linearize the Equations of Motion around an agreed linearization point.

2.2 Linearizing the Equations of Motion

A nonlinear system should be linearized in order to make the forthcoming calculations easier in terms of analyzing any derived differential equations. For the MinSeg, a small signal analysis method of linearization is adequate, because the control system needs to be stable only in the neighborhood of the equilibrium [17, Section 9.2.1]. Thus the key to linearize the Equations of Motion is to find where the plant is at equilibrium. Despite the apparent control challenges, if the equilibrium for the linearization point is found, in return the linearized system control law equation designs are easier to use, calculate with and describe mathematically, than any nonlinearized system designs [48, Section 2.6]. With good implementation of Taylor series [43], the nonlinear function can be traced with good accuracy and is very useful in the linearization applications, assuming the signals are continuous and the nonlinearities of the function are smooth. Also the nonlinear relationship to the state itself (e.g. wheel acceleration) and/or the control is required to be able to compute a valid linear model. Thus, the differential equations can't be written as

$$\dot{\mathbf{x}}=\mathbf{A}\mathbf{x}+\mathbf{B}u,$$

but are left in the form of

$$\dot{\mathbf{x}}=\mathbf{f}(\mathbf{x},u)$$

[17, Section 9.2.1]. Because of this, one cannot produce the matrices of a State-Space by nonlinear functions. The linearization can be done in two steps [48, Section 2.6]:

1. Compute the equilibrium point (and if applicable, use the Taylor series).
2. Then compute the matrices.

The nonlinear terms of the derived EOM in equation (30) are $\sin(\theta_b)$, $\ddot{x}_w\cos(\theta_b)$, $\ddot{\theta}_b\cos(\theta_b)$ and $\dot{\theta}_b^2\sin(\theta_b)$. Alas they need to be linearized and as expected, the linearization point will be the equilibrium, where the body angle is zero ($\theta_b = 0$) as the robot will be standing almost exactly straight up. One can imagine that the MinSeg will fall if not set up straight to the equilibrium point, so the nonlinear term values are set to zero as follows:

$$sin(\theta_b)\approx\theta_b,\quad \ddot{x}_w\cos(\theta_b)\approx\ddot{x}_w,\quad \ddot{\theta}_b\cos(\theta_b)\approx\ddot{\theta}_b\quad and\quad \dot{\theta}_b^2\approx 0.$$

The centripetal forces can be assumed to be negligible (i.e. small body angle velocities) for $\dot{\theta}_b^2 sin(\theta_b)$. The linearized EOM are presented in (31). [49, Section 3.2]

$$\begin{cases}\left(\frac{I_w}{l_w}+l_w m_b+l_w m_w\right)\ddot{x}_w=-m_b l_b l_w\ddot{\theta}_b+\frac{K_t}{R_m}v_m-\left(\frac{K_e K_t}{R_m}+b_f\right)\left(\frac{\dot{x}_w}{l_w}-\dot{\theta}_b\right)\\ (I_b+m_b l_b^2)\ddot{\theta}_b=+m_b l_b g\theta_b-m_b l_b\ddot{x}_w-\frac{K_t}{R_m}v_m+\left(\frac{K_e K_t}{R_m}+b_f\right)\left(\frac{\dot{x}_w}{l_w}-\dot{\theta}_b\right)\end{cases}\tag{31}$$

Comparing to (30), in (31) the terms have changed because of setting the aforementioned values to zero. So the term $-m_b l_b l_w\ddot{\theta}_b\cos(\theta_b)$ is changed to $-m_b l_b l_w\ddot{\theta}_b$ because of $\ddot{\theta}_b\cos(\theta_b)\approx\ddot{\theta}_b$. The term $+m_b l_b l_w\dot{\theta}_b^2\sin(\theta_b)$ is changed to zero and cancelled out

of the equation because of the agreed linearization condition $\dot{\theta}_b^2 \approx 0$, where the effect of the zero sets in instantly, thus the term is not carried forward. The term $+m_b l_b g \sin(\theta_b)$ is changed to $+m_b l_b g \theta_b$ because of $sin(\theta_b) \approx \theta_b$ and $\theta_b = 0$. Even though $\theta_b = 0$, the effect of zero starts only when equation (30) is run thorough MATLAB as matrices, hence we need to express the whole equation part $+m_b l_b g \sin(\theta_b)$ with the θ_b term, as it is needed when the angle of the body changes in the simulator or in a field-test. And finally the $-m_b l_b \ddot{x}_w \cos(\theta_b)$ term is changed to $-m_b l_b \ddot{x}_w$ because of $\ddot{x}_w \cos(\theta_b) \approx \ddot{x}_w$. Differential equations can be described as matrices in the State-Space form by selection of the state variables that are the most suitable for a given system [17, Section 7.2]. The state-space form of the Equations of Motion is the preferred form of matrix representation here and it suits MATLAB very well. The basic State-Space form [42] is presented in equation (32). [49, Section 3.2]

$$\begin{cases} \dot{\mathbf{x}} = A\mathbf{x} + Bu \\ y = C\mathbf{x} + Du \end{cases} \tag{32}$$

Where A is called the control matrix (or system matrix), B input matrix, C output matrix and D feedforward matrix, for which there is no direct feedthorough yet in the system, thus matrix D is not needed for the time being as we are now interested in a feedback controller for which matrices A, B and C are sufficient. The $\mathbf{x}$ is called the state vector, y the output variable and u the input variable. For the MinSeg, the input variable u is equal to the motor input voltage v_m and the output variable y is equal to the body angle θ_b [49, Section 3.3].

$$\begin{cases} u = v_m \\ y = \theta_b \end{cases} \tag{33}$$

The input and output variables are set to the ones in (33) for the time being and next the balancing system is written both in a parametric form as in (34) for mathematical convenience and later in this Chapter in a parametric numeric form, where the variable values are implemented into the matrices [49, Section 3.3].

$$\begin{bmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \end{bmatrix} \begin{bmatrix} \ddot{x}_w \\ \ddot{\theta}_b \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \end{bmatrix} \begin{bmatrix} x_w \\ \dot{x}_w \\ \theta_b \\ \dot{\theta}_b \end{bmatrix} + \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} u \tag{34}$$

In (34), the α, β and γ represent the different parameter functions, where their numerical values will be recorded. Equation (34) can also be expressed in a way such as in equation (35) [49, Section 3.3].

$$\begin{bmatrix} \ddot{x}_w \\ \ddot{\theta}_b \end{bmatrix} = \begin{bmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \end{bmatrix}^{-1} \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \end{bmatrix} \begin{bmatrix} x_w \\ \dot{x}_w \\ \theta_b \\ \dot{\theta}_b \end{bmatrix} + \begin{bmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \end{bmatrix}^{-1} \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} u \tag{35}$$

Where on the left hand side of the equation resides only the suitable functions, which are the wheel acceleration $\ddot{x}_w$ and body acceleration $\ddot{\theta}_b$, so that the resemblance is as close to the desired form of equation (32) as possible. With the help of equation (35), we can now write matrices A, B and C into the state-space form of (32) [49, Section 3.3].

$$\left(\frac{I_w}{l_w} + l_w m_b + l_w m_w\right)\ddot{x}_w = -m_b l_b l_w \ddot{\theta}_b + \frac{K_t}{R_m} v_m - \left(\frac{K_e K_t}{R_m} + b_f\right)\left(\frac{\dot{x}_w}{l_w} - \dot{\theta}_b\right) \tag{36}$$

Here, equation (36) represents the upper part of the equation (31) for $\ddot{x}_w$, and can be rewritten as in equation (37) [49, Section 3.3].

$$\gamma_{11}\ddot{x}_w + \gamma_{12}\ddot{\theta}_b = \alpha_{12}\dot{x}_w + \alpha_{14}\dot{\theta}_b + \beta_{11} v_m \tag{37}$$

In equation (37), the γ_{xx}, α_{xx} and β_{xx} parameter functions have replaced their respective counterparts of the terms in (36) and the equivalence can be seen in the equations of (38) [49, Section 3.3].

$$\begin{cases} \gamma_{11} = \left(\frac{I_w}{l_w} + l_w m_b + l_w m_w\right) \\ \gamma_{12} = +m_b l_b l_w \\ \alpha_{12} = -\left(\frac{K_e K_t}{R_m} + b_f\right)\frac{1}{l_w} \\ \alpha_{14} = +\left(\frac{K_e K_t}{R_m} + b_f\right) \\ \beta_{11} = +\frac{K_t}{R_m} \end{cases} \tag{38}$$

$$(I_b + m_b l_b^2)\ddot{\theta}_b = +m_b l_b g \theta_b - m_b l_b \ddot{x}_w - \frac{K_t}{R_m} v_m + \left(\frac{K_e K_t}{R_m} + b_f\right)\left(\frac{\dot{x}_w}{l_w} - \dot{\theta}_b\right) \tag{39}$$

Also in the same way, equation (39) represents the lower part of the equation (31) for $\ddot{\theta}_b$ and can be rewritten as in equation (40) [49, Section 3.3].

$$\gamma_{21}\ddot{x}_w + \gamma_{22}\ddot{\theta}_b = \alpha_{22}\dot{x}_w + \alpha_{23}\theta_b + \alpha_{24}\dot{\theta}_b + \beta_{21} v_m \tag{40}$$

Likewise, in equation (40), the γ_{xx}, α_{xx} and β_{xx} parameter functions have replaced their respective counterparts of the terms in (39) and the equivalence can be seen in the equations of (41) [49, Section 3.3].

$$\begin{cases} \gamma_{21} = m_b l_b \\ \gamma_{22} = (I_b + m_b l_b^2) \\ \alpha_{22} = \left(\frac{K_e K_t}{R_m} + b_f\right)\frac{1}{l_w} \\ \alpha_{23} = m_b l_b g \\ \alpha_{24} = -\left(\frac{K_e K_t}{R_m} + b_f\right) \\ \beta_{21} = -\frac{K_t}{R_m} \end{cases} \tag{41}$$

Notice that the variables α_{11}, α_{13} have no place and no value in equation (36) and α_{21} has no place and no value in equation (39). Thus, in the absence of those variables, one can write the equation (34) again with zeros replacing those null variables and rewrite the EOM as in (42) [49, Section 3.3].

$$\begin{bmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \end{bmatrix} \begin{bmatrix} \ddot{x}_w \\ \ddot{\theta}_b \end{bmatrix} = \begin{bmatrix} 0 & \alpha_{12} & 0 & \alpha_{14} \\ 0 & \alpha_{22} & \alpha_{23} & \alpha_{24} \end{bmatrix} \begin{bmatrix} x_w \\ \dot{x}_w \\ \theta_b \\ \dot{\theta}_b \end{bmatrix} + \begin{bmatrix} \beta_{11} \\ \beta_{21} \end{bmatrix} u \tag{42}$$

Because of the fact that the robot will stay still at the equilibrium point and its wheel is not moving, the term x_w is zero regarding to the parameter function α_{11}. For the same reason α_{21} is zero, as the angle θ_b is not changing from zero at the equilibrium point. The suitable variable is selected as $\ddot{x}_w$ and thus is already in use for γ_{11}, so α_{13} is also zero [49, Section 3.3]. In formula (43), the outcome of a matrix inverse is presented. The matrix inverse is the way how matrices are divided. In matrices, direct dividing is not applicable, so dividing is done by multiplying the matrix with its inverse (i.e. reciprocal of a number) [22]. In equation (35), the matrix (34) was presented in an alternative way, where the parts changed places from the left hand side of the equation:

$$\begin{bmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \end{bmatrix}$$

to the right:

$$\begin{bmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \end{bmatrix}^{-1}.$$

This is where an inverse of a matrix steps in and the result can be seen in equation (43) [49, Section 3.3].

$$\begin{bmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \end{bmatrix}^{-1} = \frac{1}{\gamma_{11}\gamma_{22} - \gamma_{12}\gamma_{21}} \begin{bmatrix} \gamma_{22} & -\gamma_{12} \\ -\gamma_{21} & \gamma_{11} \end{bmatrix} \tag{43}$$

With the accumulated knowledge of the system, two new matrices ((44) and (45)) can be defined [49, Section 3.3].

$$\begin{bmatrix} 0 & a_{22} & a_{23} & a_{24} \\ 0 & a_{42} & a_{43} & a_{44} \end{bmatrix} := \frac{1}{\gamma_{11}\gamma_{22} - \gamma_{12}\gamma_{21}} \begin{bmatrix} \gamma_{22} & -\gamma_{12} \\ -\gamma_{21} & \gamma_{11} \end{bmatrix} \begin{bmatrix} 0 & \alpha_{12} & 0 & \alpha_{14} \\ 0 & \alpha_{22} & \alpha_{23} & \alpha_{24} \end{bmatrix} \tag{44}$$

$$\begin{bmatrix} b_{21} \\ b_{41} \end{bmatrix} := \frac{1}{\gamma_{11}\gamma_{22} - \gamma_{12}\gamma_{21}} \begin{bmatrix} \gamma_{22} & -\gamma_{12} \\ -\gamma_{21} & \gamma_{11} \end{bmatrix} \begin{bmatrix} \beta_{11} \\ \beta_{21} \end{bmatrix} \tag{45}$$

Because of the three null variables in equation (42), equation (44) inherited the same null variables. For the new a_{xx} and b_{xx} variables that were chosen arbitrarily into the two new matrices (44) and (45), matrix multiplication rules [20] were used to define their contents. As $\gamma_{11}\gamma_{22} - \gamma_{12}\gamma_{21}$ can be set to be δ, then

$$\frac{1}{\gamma_{11}\gamma_{22} - \gamma_{12}\gamma_{21}} = \frac{1}{\delta}.$$

In the matrix (44): For a_{22}, the multiplications were

$$\frac{1}{\delta} \times \gamma_{22}\alpha_{12} + \frac{1}{\delta} \times (-\gamma_{12}\alpha_{22}).$$

For a_{23}, the multiplications were

$$\frac{1}{\delta} \times \gamma_{22} \times 0 + \frac{1}{\delta} \times (-\gamma_{12}\alpha_{23}).$$

For a_{24}, the multiplications were

$$\frac{1}{\delta} \times \gamma_{22}\alpha_{14} + \frac{1}{\delta} \times (-\gamma_{12}\alpha_{24}).$$

For a_{42}, the multiplications were

$$\frac{1}{\delta} \times (-\gamma_{21}\alpha_{12}) + \frac{1}{\delta} \times \gamma_{11}\alpha_{22}.$$

For a_{43}, the multiplications were

$$\frac{1}{\delta} \times (-\gamma_{21}) \times 0 + \frac{1}{\delta} \times \gamma_{11}\alpha_{23}.$$

For a_{44}, the multiplications were

$$\frac{1}{\delta} \times (-\gamma_{21}\alpha_{14}) + \frac{1}{\delta} \times \gamma_{11}\alpha_{24}.$$

For b_{21}, the multiplications were

$$\frac{1}{\delta} \times \gamma_{22}\beta_{11} + \frac{1}{\delta} \times (-\gamma_{12}\beta_{21}).$$

And for b_{41}, the multiplications were

$$\frac{1}{\delta} \times (-\gamma_{21}\beta_{11}) + \frac{1}{\delta} \times \gamma_{11}\beta_{21}.$$

To sum up the multiplied variables into a clear list, the variable equation list (46) was created. [49, Section 3.3]

$$\begin{cases} \delta = \gamma_{11}\gamma_{22} - \gamma_{12}\gamma_{21} \\ a_{22} = \frac{\gamma_{22}\alpha_{12} - \gamma_{12}\alpha_{22}}{\delta} \\ a_{23} = \frac{-\gamma_{12}\alpha_{23}}{\delta} \\ a_{24} = \frac{\gamma_{22}\alpha_{14} - \gamma_{12}\alpha_{24}}{\delta} \\ b_{21} = \frac{\gamma_{22}\beta_{11} - \gamma_{12}\beta_{21}}{\delta} \\ a_{42} = \frac{-\gamma_{21}\alpha_{12} + \gamma_{11}\alpha_{22}}{\delta} \\ a_{43} = \frac{+\gamma_{11}\alpha_{23}}{\delta} \\ a_{44} = \frac{-\gamma_{21}\alpha_{14} + \gamma_{11}\alpha_{24}}{\delta} \\ b_{41} = \frac{-\gamma_{21}\beta_{11} + \gamma_{11}\beta_{21}}{\delta} \end{cases} \tag{46}$$

Now the rewritten system looks like this (47) [49, Section 3.3].

$$\begin{cases} \begin{bmatrix} \dot{x}_w \\ \ddot{x}_w \\ \dot{\theta}_b \\ \ddot{\theta}_b \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & 0 & 1 \\ 0 & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_w \\ \dot{x}_w \\ \theta_b \\ \dot{\theta}_b \end{bmatrix} + \begin{bmatrix} 0 \\ b_{21} \\ 0 \\ b_{41} \end{bmatrix} u \\ y = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_w \\ \dot{x}_w \\ \theta_b \\ \dot{\theta}_b \end{bmatrix} \end{cases} \tag{47}$$

In equation (47), the first row is

$$\begin{bmatrix} \dot{x}_w \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x}_w \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} u$$

and produces

$$\begin{bmatrix} \dot{x}_w \end{bmatrix} = \begin{bmatrix} \dot{x}_w \end{bmatrix},$$

which is the opportune function

$$\begin{bmatrix} \dot{x}_w \end{bmatrix}$$

itself. The second row is

$$\begin{bmatrix} \ddot{x}_w \end{bmatrix} = \begin{bmatrix} 0 & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} \dot{x}_w \end{bmatrix} + \begin{bmatrix} b_{21} \end{bmatrix} u$$

and produces

$$\begin{bmatrix} \dot{x}_w \end{bmatrix} = \begin{bmatrix} 0x_w & a_{22}\dot{x}_w & a_{23}\theta_b & a_{24}\dot{\theta}_b \end{bmatrix} + \begin{bmatrix} b_{21} \end{bmatrix} u,$$

which is the same as the second row. The third row is

$$\begin{bmatrix} \dot{\theta}_b \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta}_b \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} u$$

and produces

$$\begin{bmatrix} \dot{\theta}_b \end{bmatrix} = \begin{bmatrix} \dot{\theta}_b \end{bmatrix},$$

which is the opportune function

$$\begin{bmatrix} \dot{\theta}_b \end{bmatrix}$$

itself. The fourth row is

$$\begin{bmatrix} \ddot{\theta}_b \end{bmatrix} = \begin{bmatrix} 0 & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} \dot{\theta}_b \end{bmatrix} + \begin{bmatrix} b_{41} \end{bmatrix} u$$

and produces

$$\begin{bmatrix} \ddot{\theta}_b \end{bmatrix} = \begin{bmatrix} 0x_w & a_{42}\dot{x}_w & a_{43}\theta_b & a_{44}\dot{\theta}_b \end{bmatrix} + \begin{bmatrix} b_{41} \end{bmatrix} u,$$

which is the same as the fourth row. For the output y, the stored variable is θ_b. It is natural to choose x_w, $\dot{x}_w$, θ_b and $\dot{\theta}_b$ as the system states, so therefore in equation (47), there are four equations, so that the aforementioned condition of the states is fulfilled. When the codes [46] are downloaded and implemented in MATLAB for the

matrix (47) according to the MinSeg Datasheet [50], the following values in (48) are obtained.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -435.0 & -6.1 & 9.1 \\ 0 & 0 & 0 & 1 \\ 0 & 1903.4 & 62.0 & -40.0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 20.5759 \\ 0 \\ -90.0275 \end{bmatrix} \tag{48}$$

The values in the matrices A and B are related to the pole and zero locations of the transfer function, which are explained more thoroughly in the next Chapters. The parameters should be almost equal depending on MATLAB itself to those ones presented in the matrices of (48), which is obtained by running the file `LabA_Solutions_LoadStateSpaceMatrices.m`. The physical attribute chart regarding their reported and measured values of the robot assembly can be seen in the corresponding table here.

Attribute	Reported value	Measured value
Total robot mass	0.381 kg (with batteries)	From 0.380 kg to 0.382 kg
Two wheel's mass	0.036 kg	0.026 kg

According to the robot's Datasheet [50], the physical attributes were found to differ from the measured ones. The weight measurements were not done in laboratory conditions or with precise equipment, but with a regular household digital scale (Camry EK3052), which had 0.002 kg accuracy. Because of the poor accuracy, no exact measurements could be performed. Thus the reported values were used also in the simulator and field-testing. But a noticeable difference was discovered with the two wheel's mass. A 0.010 kg weight difference, even if the scale's accuracy (0.024 - 0.028 kg) is taken into account, is quite large. The robot assembly's overall weight was measured almost exactly the same as the reported one, being 0.381 kg. There was no explanation found for the weight difference. An approximately 13 cm of clear Scotch tape was used to attach to the battery pack to hold it in its place, which was measured to be 0.000 kg, as tape is very light. The tape usage is adviced because the batteries are easy to fall off from the battery deposit box behind the robot's Arduino board. There were six Sanyo Eneloop 1.2 V recharchable Ni-Mh AA-batteries used and even though different batteries of AA size can have a few grams different weight, the measured weight with these batteries on the robot gave oddly the same total weight as the ones reported in the robot's Datasheet. A model and fabricator of any batteries recommended to be used with the MinSeg robot's assembly is not known. The USB cable was not attached to the robot while the scale measurements were performed and is not supposed to be attached to the robot while it is balancing itself, but is only used as the serial port and thus a medium of transporting the data to the robot's internal memory. The wheel shaft is a Lego piece that goes thorough the motor wheel shaft hole and is what keeps the wheels attached to the motor. The wheel's width measured at the widest point was kept at 7.0 cm, which is subject to change if the wheels and/or the wheel shaft is set slightly off the motor center point or too close to or set further apart from the motor. There is a few millimeter tolerance in the MinSeg for setting up the wheel disposition.

2.3 Determining the Transfer Function

To get the transfer function, one needs to take the Laplace transform [17, Section 3.1.7] of the system's unit impulse response. The system characteristic equation, which is used to determine the system characteristics, would thus be the inverse of the transfer function and equal to the unit impulse response. The frequency response, which is the system response to a sinusoid, can be found by using the exponential response of a unit impulse that results from finding the transfer function [17, Section 3.1.2]. If a transfer function has a sinusoidal form of a complex exponential as the input signal, it outputs the same sinusoid signal multiplied by the amplified value of the transfer function added with a phase delay related to the transfer function. So the output of a transfer function will be another sinusoid of the same frequency, amplified and shifted by certain factors, which come from the transfer function at the input frequency. Thus the physical meaning of a transfer function is how the corresponding system responds to an arbitrary sinusoid [48, Section 2.10]. A Bode plot, discussed in Chapter 3.2.3, is a graphical representation of both the amplitude and the phase delay of the transfer function. An impulse response is associated to the transfer function both in continuous s-plane (by the Laplace transform) and discrete z-plane (by the Z-transform) time domains. The $G(s)$ refers to the transfer function in continuous-time and $G(z)$ refers to the transfer function in discrete-time domain. The concept of gain is used in automatic control to determine system response. Gain can be determined from a transfer function by setting up the input signal as an unitary step (the impulse response) and then waiting for the system to respond and determining the stabilized value for which the system output has reached [48, Section 2.10]. Next we will compute the system's transfer function and also it's poles and zeros. The poles are the roots of the denominator and locate in the s-plane where the TF becomes infinite. The zeros dampen the effects of the poles by blocking the coinciding frequencies of the poles in close proximity regions [17, Section 3.1.8]. The s-plane is the complex plane where the Laplace transforms reside [34]. The effect on pole locations [17, Section 3.3] on a system is an essential factor of its characteristics and an experienced designer can determine a lot from a system just by looking at its Pole-Zero Map. But some characteristics of a system, like its controllability, cannot be known only by looking at the Pole-Zero Map or either from the transfer function, but must be simulated and tested [17, Section 7.4]. The transfer function can be obtained from the State-Space representation in the following way. From the State-Space representation in the time domain

$$\begin{cases} \dot{\mathbf{x}}(t) = A\mathbf{x}(t) + Bu(t) \\ y(t) = C\mathbf{x}(t) + Du(t) \end{cases}$$

the Laplace transform

$$\begin{cases} s\mathbf{X}(s) - \mathbf{x}(0) &= \mathbf{A}\mathbf{X}(s) + \mathbf{B}U(s) \\ Y(s) &= \mathbf{C}\mathbf{X}(s) + \mathbf{D}U(s) \end{cases}$$

in the s domain is obtained and as an algebraic form, is relatively easy to handle. Here, the $\mathbf{X}(s)$ means a vector, which has the components that are the Laplace

transforms of the state variables and where the initial values can be changed to 0 ($\mathbf{x}(0) = 0$). Now the first line of the State-Space representation takes the form of

$$s\mathbf{X}(s) \quad = \quad \mathbf{A}\mathbf{X}(s) + \mathbf{B}U(s),$$

which in turn is

$$[sI - A]\mathbf{X}(s) \quad = \quad \mathbf{B}U(s),$$

where the s is a scalar value, A is a square matrix and I is an identity matrix of the same dimension. In simple terms, this is just a group of equations that are represented by matrices. The identity matrix is necessary to be included into the calculations, because the subraction between the scalar s and the matrix A is not defined otherwise. From the previous equation, $\mathbf{X}(s)$ can be solved by multiplying the equation from the left hand side with the inverse matrix of $[sI - A]$ like this

$$\begin{aligned}(sI - A)^{-1}(sI - A)\mathbf{X}(s) \quad &= \quad (sI - A)^{-1}\mathbf{B}U(s) \\ I\mathbf{X}(s) \quad = \quad \mathbf{X}(s) \quad &= \quad (sI - A)^{-1}\mathbf{B}U(s).\end{aligned}$$

Now that the state vector $\mathbf{X}(s)$ is eliminated, the result of the previous equation can be inserted into the measurement equation

$$Y(s) \quad = \quad \mathbf{C}\mathbf{X}(s) + \mathbf{D}U(s) \quad = \quad \mathbf{C}(sI - A)^{-1}\mathbf{B}U(s) + \mathbf{D}U(s).$$

Thus the transfer function, which is the relation between output y and input u, is obtained from

$$G(s) = \frac{Y(s)}{U(s)} \quad = \quad \mathbf{C}(sI - A)^{-1}\mathbf{B}U(s) + \mathbf{D}U(s).$$

The transfer function can also be obtained from the matrix form by using the following formula

$$G(s) = \frac{det\begin{bmatrix} sI - A & -B \\ C & D \end{bmatrix}}{det(sI - A)}.$$

[17, Section 7.4.2] [37] The matrix formulas presented by (49) are the produce of the combination of matrices A, B and C from (47) (and the absent D matrix) added to the identity matrix I [28] that has been multiplied by an s term. The identity matrix is there to help make the calculations easier to handle and is similar to multiplying a number by the term 1, which won't change the outcome of the multiplication. By analyzing the matrices in (49), we can see which poles and zeros are to be found [49, Section 3.4].

$$sI - A \qquad and \qquad \begin{bmatrix} sI - A & -B \\ C & D \end{bmatrix} \tag{49}$$

The transfer function should be expressed in the form of (50) [49, Section 3.4], where K represents the gain factor, z the zero quantity and p the pole quantity. The

reason why all the zeros are never on the denominator is because dividing by zero is undefined [14].

$$G(s) = K\frac{\prod_i(s - z_i)}{\prod_j(s - p_j)} \tag{50}$$

The system poles (or in the zeros' case the zeros), written here as s where the matrix is singular, can be seen in the matrix (51). A singular value is a non-negative real number [38] and when a pole s is 0, then the corresponding matrix for those poles is singular, thus a pole can be 0 [49, Section 3.4]. When the transfer function is transformed into state-space, the result is seen in (51) [17, Section 7.4].

$$sI - A = \begin{bmatrix} s & -1 & 0 & 0 \\ 0 & s - a_{22} & -a_{23} & -a_{24} \\ 0 & 0 & s & -1 \\ 0 & -a_{42} & -a_{43} & s - a_{44} \end{bmatrix} \tag{51}$$

The zeros of the system, written here also as s, can be seen in the matrix (52).

$$\begin{bmatrix} sI - A & -B \\ C & D \end{bmatrix} = \begin{bmatrix} s & -1 & 0 & 0 & 0 \\ 0 & s - a_{22} & -a_{23} & -a_{24} & -b_2 \\ 0 & 0 & s & -1 & 0 \\ 0 & -a_{42} & -a_{43} & s - a_{44} & -b_4 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{52}$$

Here we can see that a 0 is zero in the matrix (at imaginary $-b_5$), thus it will be cancelled out in the corresponding TF [49, Section 3.4]. The reason why the terms $-b_2$ and $-b_4$ are not called $-b_{21}$ and $-b_{41}$ as in (47) is that later in equation (68) we don't want to mix these different terms in MATLAB, even though they still represent the same terms respectfully. As there is one cancellation (the D matrix) in the TF, there will be an unobservable state in the transfer function [49, Section 3.4]. Now that the matrices have been calculated analytically for the poles and the zeros, they can be inserted into MATLAB and calculated numerically. The resulting TF can be seen in equation (53). The TF of (53) can be obtained after running `LabA_Solutions_LoadPhysicalParameters.m`, `LabA_Solutions_LoadStateSpaceMatrices.m` (and later `LabA_Solutions_ComputePIDGains.m`) and lastly the file `LabA_Solutions_ComputeTransferFunction.m` from [46] in this order in MATLAB [49, Section 3.4].

$$G(s) = \frac{-90.03s^2 - 1.053 \times 10^{-11}s - 1.707 \times 10^{-26}}{s^4 + 475s^3 - 62.02s^2 - 1.537 \times 10^4 s} \approx \frac{-90s}{(s + 475.0)(s + 5.7)(s - 5.7)} \tag{53}$$

The transfer function shows four false zeros [49, Section 3.4] in the numerator and one in the denominator, which can be presented as a product of 1.0e+04:

$$\begin{bmatrix} 0 & 0 & -90.0275 & 0 & 0 \\ 0.0001 & 0.0475 & -0.0062 & -1.5371 & 0 \end{bmatrix}.$$

In (53), the terms besides the $-90s$ in the numerator, are false, and the term $-1.053 \times 10^{-11}s$ is the produce of the spurious zero from the imaginary term $-b_{51}$.

The false terms have to be removed and the cleaned up fraction is seen on the right hand side of equation (53). The Pole-Zero plot of Figure 8, is then derived after running the file `LabA_Solutions_LoadPhysicalParameters.m` [49, Section 3.4].

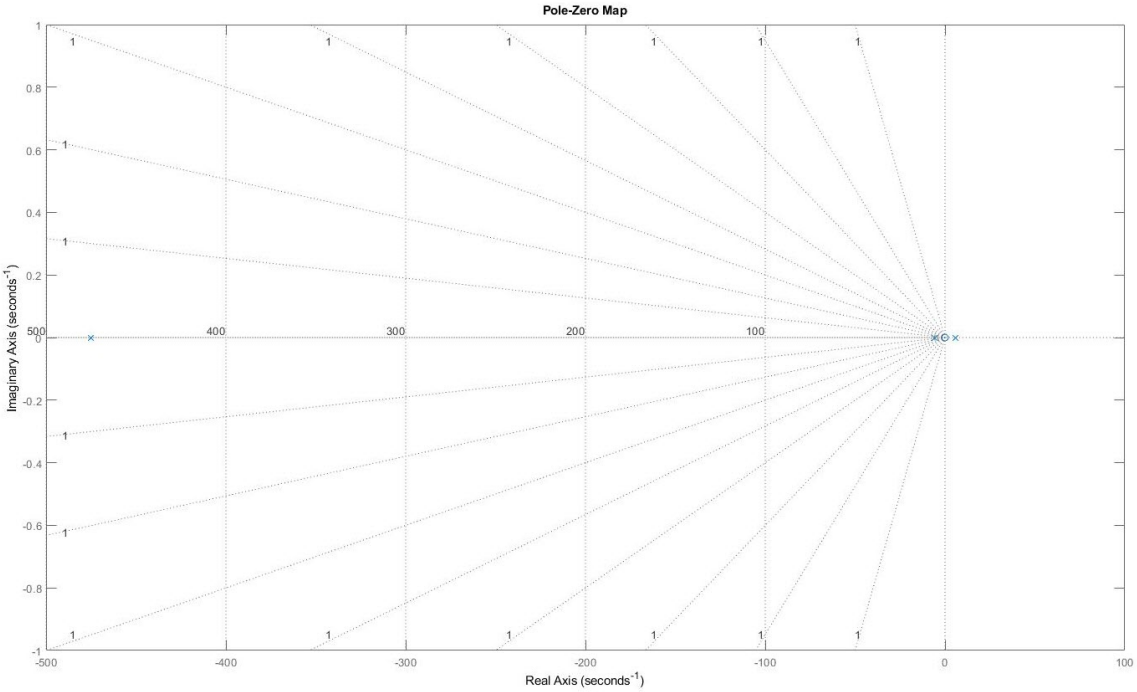

Figure 8: The Pole-Zero Map of the linearized system of (47) with parameters from (48) [49, Section 3.4].

In Figure 8, we can see the poles and the zeros of the system and also notice that the system is unstable [8]. The reason for this is that there is a pole, which is represented by the blue X symbol, on the right-half of the imaginary axis (the vertical 0-axis) of the s-plane (the Pole-Zero Map). At the origin, there are two poles side-by-side along with a zero in between. In addition to these, there is one pole on the left-half plane (LHP) [34] at -475.0 on the real axis. Thus we have three poles and one zero in the system as suggested in the TF (53) [49, Section 3.4].

3 Applying Theory to Practice

3.1 Introduction

Now it is time to gather all the acquired mathematical knowledge into a real system and start controlling and tuning the actual robot. Before any practical experiments are initiated, we must know how to use MATLAB along with Simulink and find a testing area as close as possible to the specified conditions. Technically, when the PID has been designed and with it the transfer function stabilized, the disturbance modelling for the robot can begin. Then everything should be in working order for converting the PID to descrete-time domain. As the system has been discretized, the closed-loop system can be simulated and it's performance tuned in Simulink. After the robot simulations have been completed, the field-testing will come next. The field-testing of the PID in real life can commence since the robot has been fully assembled, communications successfully established and the program uploaded into the Arduino-microcontroller [49, Section 3.5].

3.2 The PID Controller

3.2.1 Stabilizing the Transfer Function

Now, we would like to stabilize the TF from the input signal of the motor voltage v_m to the output signal of the body angle θ_b [49, Section 3.5].

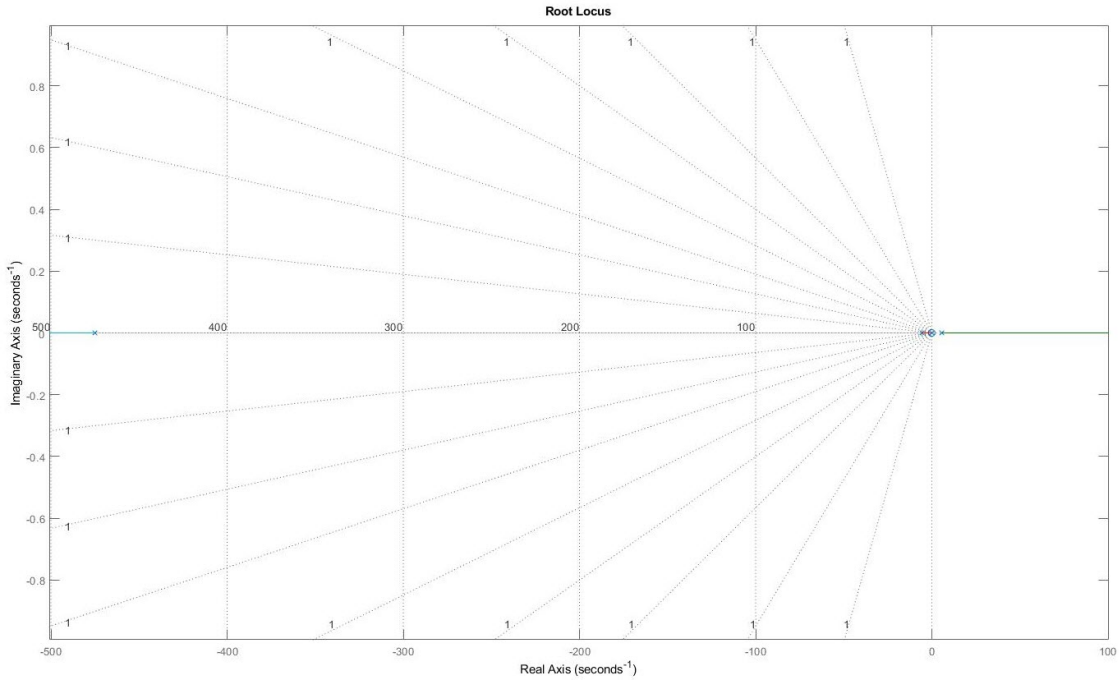

Figure 9: The root locus of the linearized system of (47) with parameters from (48) [49, Section 3.5].

An unstable transfer function will make the robot controls fail as the control signal will start to wander out of the stabilized range. To this purpose, a PID [17, Section

4.3] needs to be designed that does not make the robot fall. The stable region is near the equilibrium, which in turn is the linearization point where the reference signal for the body angle θ_b is zero. There the robot will stand straight up. The wheel position is of no importance right now, so x_w is ignored. By looking at the root locus [21] of the linearized system in Figure 9, there is no signal path between the poles and the zero. This means that there is no proportional gain-term P that stabilizes the system, so trying to control the robot with only a P controller will not be adequate. Thus there is no P-controller stabilizing θ_b [49, Section 3.5]. As can be seen in equation (61), with the highest order of the denominator is three for the poles, there are three poles in the system [21]. None of the poles are connecting to the zero with the current P controller, so an integral term should be added alongside a derivative term. The derivative term will expand the stability domain compared to just a PI-controller, so a PID controller is the best choice [49, Section 3.5]. The PID is designed with the poles allocation method [48, Section 3.6]. After the closed-loop is stable, the poles can be placed as desired. The stability is reached after the unstable pole from $+5.7$ (the $(s-5.7)$ in (53)) is removed and placed into -3 along the real axis. When the dominant second order pole placement system, or the symmetric root locus is used in designing the initial pole placement, some tweaking of the pole placement usually takes place after the method has been selected. More about those methods are discussed here in Chapter 4.2.1 [17, Section 7.6.3]. If the system is controllable in s domain, the poles can be placed anywhere that is desired on the LHP [7], but more about controllability is explained in Chapter 4.1. Here, as no preference of pole placement existed initially, other than a place where the pole didn't interfere with an already proven concept, the reference position for the pole of -3 along the real axis was selected [49, Section 3.5]. The dominant second-order pole placement method that was used in the reference, selected the pole of -3 probably because it is in the middle between the other stable pole of -5.7 and the origin, where the pole -3 doesn't interfere with the functioning system dynamics by being too close to the origin and thus the imaginary axis and the unstable RHP and not too close to another pole which might also interfere the positive effects of a dominant pole by underdampening the system. [17, Section 7.6.1] We could move the pole further from the origin for a better system response in theory, but if taken to an extreme, in practice this could stress the actuators to a breaking point. To accidentally or seldom exit the linearized zone in real life in the MinSeg's case is not very dangerous. The actuator in this case would be the electric motor of the robot. Other poles and the zero can stay where they are [49, Section 3.5]. The problem of stability of a system that is observed from the s-plane exists only when there is a pole on the right half-plane (RHP) [34] of the imaginary axis from the origin. If a solitary zero or zeros reside on the right half-plane or an equal amount of poles and zeros reside on the RHP, it does not make the system unstable. Only when there are more poles than the zeros there, it becomes a problem. The zeros' effect to poles is usually dampening, but in the case of a zero being at close proximity to a pole, the effect on the whole system is making the plant controls close to impossible [17, Section 7.6].

$$C(s) = k_P + k_I\frac{1}{s} + k_D s = \frac{k_P s + k_I + k_D s^2}{s} = \frac{N_C}{D_C} \tag{54}$$

In (54), we can see the PID controller equation, where the P, I and D terms are combined together into a fraction. The transfer function, on the other hand, is represented in equation (55). In both equations, the N stands for the numerator and D the denominator. The footnote $_C$ and $_G$ represent the controller and the transfer function portions respectfully. The kappa κ here is an alternative notation for gain, as we have two gain factors, where κ is for the transfer function and k is for the PID [49, Section 3.5].

$$G(s) = \frac{\kappa s}{(s-p_1)(s-p_2)(s-p_3)} = \frac{N_G}{D_G} \tag{55}$$

When the equation for the controller (54) is combined with the equation for the transfer function (55), the resulting closed-loop system will be such as in equation (56) [17, Section 4.1] [49, Section 3.5].

$$\begin{aligned} P(s) = \frac{CG}{1+CG} &= \frac{\frac{N_C}{D_C}\frac{N_G}{D_G}}{1+\frac{N_C}{D_C}\frac{N_G}{D_G}} = \frac{N_C N_G}{N_C N_G + D_C D_G} \\ &= \frac{\kappa s(k_P s + k_I + k_D s^2)}{\kappa s(k_P s + k_I + k_D s^2) + s(s-p_1)(s-p_2)(s-p_3)} \\ &= \frac{\kappa k_D s^3 + \kappa k_P s^2 + \kappa k_I s}{\kappa k_D s^3 + \kappa k_P s^2 + \kappa k_I s + s(s^3 - s^2(p_1+p_2+p_3) + s(p_1p_2+p_1p_3+p_2p_3) - p_1p_2p_3)} \\ &= \frac{\kappa k_D s^3 + \kappa k_P s^2 + \kappa k_I s}{s(s^3 + s^2(-p_1-p_2-p_3+\kappa k_D) + s(p_1p_2+p_1p_3+p_2p_3+\kappa k_P) + (-p_1p_2p_3+\kappa k_I))} \end{aligned} \tag{56}$$

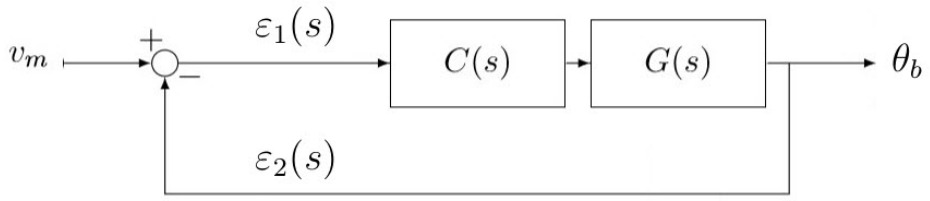

Figure 10: The closed-loop transfer function of the robot system [17, Section 3.2.1].

Figure 10 presents where the calculations of (56) are inherited from. The first fraction of (56) in the closed-loop system $P(s)$ is derived from the input

$$u = v_m$$

to the output

$$y = \theta_b$$

with the help of operators $\varepsilon_1(s)$ and $\varepsilon_2(s)$. The derivation is produced by

$$Y(s) = \theta_b \ = C(s)G(s)\varepsilon_1(s)$$

and

$$U(s) = v_m,$$

where $\varepsilon_1(s)$ is $U(s) - \varepsilon_2(s)$. Then

$$Y(s) = C(s)G(s)(U(s) - Y(s))$$

and

$$\varepsilon_2(s) = Y(s).$$

When multiplied thorough, $Y(s)$ becomes $C(s)G(s)U(s) - C(s)G(s)Y(s)$, which can be factored into

$$(C(s)G(s) + 1)Y(s) = C(s)G(s)U(s).$$

This can be arranged to

$$Y(s) = \frac{C(s)G(s)}{1 + C(s)G(s)}U(s) = G_{TOT}(s)U(s).$$

And finally to the form [6] of

$$P(s) = \frac{C(s)G(s)}{1 + C(s)G(s)}.$$

The aim is to have such a denominator as in (57) [49, Section 3.5].

$$s(s-p_1^{'})(s-p_2^{'})(s-p_3^{'}) = s(s-p_1)(s-p_2)(s-p_3^{'}) = s(s-475.0)(s-5.7)(s-3) \quad (57)$$

The reason for this is that the newly designed PID transfer function has to be equal to the old transfer function of (53) with the difference that the one positive pole is changed to a negative. Such a change is expressed as a prime ($'$) above the corresponding pole, which in this case is the $p_3^{'}$ that corresponds to the -5.7 pole in (53) and represents the next state of the system. The equating denominators of (56) and (57) can be seen from (58) [49, Section 3.5].

$$\begin{cases} -p_1 - p_2 - p_3 + \kappa k_D = -p_1 - p_2 - p_3^{'} \\ p_1p_2 + p_1p_3 + p_2p_3 + \kappa k_P = p_1p_2 + p_1p_3^{'} + p_2p_3^{'} \\ -p_1p_2p_3 + \kappa k_I = -p_1p_2p_3^{'} \end{cases} \quad (58)$$

And when (58) is cleaned up with regards to k_P, k_I and k_D, it reaches the form of (59) [49, Section 3.5].

$$\begin{cases} k_P = \frac{p_1p_3^{'} + p_2p_3^{'} - p_1p_3 - p_2p_3}{\kappa} \\ k_I = \frac{p_1p_2p_3 + p_1p_2p_3^{'}}{\kappa} \\ k_D = \frac{p_3 - p_3^{'}}{\kappa} \end{cases} \quad (59)$$

When the code [46] regarding (59) is run in MATLAB with the robot parameters [50], the resulting gains can be seen in (60) [49, Section 3.5].

$$\begin{cases} k_P = -46.5603 \\ k_I = -260.2962 \\ k_D = -0.0969 \end{cases} \quad (60)$$

By intuition, one might argue that the negative values are a result of an unstable system trying to stabilize itself with negative gains. Normally, with a stable system, these gains should be positive and no negative compensation factor of k_P is needed. Compared to the reference values [50], the results differ only on the proportional part from its -260.4. Other values were similar when rounded up to a decimal accuracy. These kinds of calculations are best done on a computer for the precision and convenience aspect, even if there are some differences on different computer systems results when running calculations on MATLAB. The differences relate to the finiteness of numerical values that are computed and compiled in various computer operating systems [49, Section 3.5].

$$P(s) = \frac{-90s^2 + 1.093 \times 10^{-11} s}{s(s^3 + 483.7s^2 + 4130s + 8063 + 5.35 \times 10^{-10})}$$

$$= \frac{-90s}{(s + 475.1)(s + 5.657)(s + 3)} \tag{61}$$

The proportional gains can be seen in (61). The spurious zero and pole were extracted in the cleaned up result from the resulting equation in (61). The reference values again differed to the magnitude of .1 for the poles, but probably because of some typo, all the poles in the reference closed-loop system transfer function $P(s)$ were marked positive [49, Section 3.5].

3.2.2 Modelling Disturbance

Disturbance modelling is useful in practical applications because there almost always will be disturbances that affect the operations of the plant [49, Section 3.6].

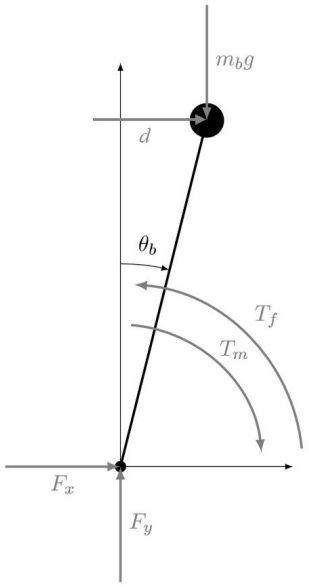

Figure 11: Disturbance d summed with F_x [49, Section 3.6].

The most useful disturbance effect to model would be a finger poking the robot, which is translated into an impulsive horizontal force that is applied to the center of mass of the robot's body. Figure 11 shows the disturbance signal as d directed towards to the body mass center (the big black dot) from the left hand side. The d is summed to the horizontal wheel-body tension component F_x as they both are horizontal forces. We must now find the linearized version with the added disturbance component (the added input d) of the EOM's that have been derived already and turn them into state-space. There is also the other input, which is the motor input voltage v_m that is different from the input d, as they enter the system state in a different angle. The poking does not disturb motor voltage and the two inputs have different dedicated columns in the B matrix and are linearly independent. To implement the effects of the poking, the equations of (30) have to be modified accordingly with the disturbance component. To begin with, we can modify equations (9) and (10). [49, Section 3.6]

$$F_x = m_b\ddot{x}_b - d \tag{62}$$

In (62), the $-d$ was added to equation (9) and the resulting F_x is now ready to be implemented into the upcoming equations. Equation (10) can be rearranged with respect to F_y like this

$$F_y = m_b\ddot{y}_b + m_b g.$$

When (9) and (10) are input into (11), the resulting equation can be seen in (63) [49, Section 3.6].

$$\begin{gathered} T_f - T_m + F_y l_b \sin(\theta_b) - F_x l_b \cos(\theta_b) = I_b\ddot{\theta}_b = \\ T_f - T_m + m_b l_b g \sin(\theta_b) + m_b l_b \ddot{y}_b \sin(\theta_b) - m_b l_b \ddot{x}_b \cos(\theta_b) + l_b \cos(\theta_b) d \end{gathered} \tag{63}$$

Where the F_y and F_x terms were replaced by $+m_b l_b g \sin(\theta_b) + m_b l_b \ddot{y}_b \sin(\theta_b)$ and $-m_b l_b \ddot{x}_b \cos(\theta_b) + l_b \cos(\theta_b) d$ respectfully. Notice here the $+l_b \cos(\theta_b) d$ component with the disturbance d that we will use next in equation (64) from the body part of equation (30) [49, Section 3.6].

$$(I_b + m_b l_b^2)\ddot{\theta}_b = +m_b l_b g \sin(\theta_b) - m_b l_b \ddot{x}_w \cos(\theta_b) - T_m + T_f + l_b \cos(\theta_b) d \tag{64}$$

In equation (64), the disturbance component from (63) is added while the rest of the equation stays the same as in (30) [49, Section 3.6]. The reason why the unlinearized equation (30) is used here instead of the linearized version (31) is that the linearization can only be done after having the disturbance component added first into the unlinearized version. In (65), the equation is exactly the same as equation (20), but with the added component of $-d$ from (62). Thus the resulting equation is (65) [49, Section 3.6].

$$F_x = m_b\ddot{x}_w + m_b l_b \ddot{\theta}_b \cos(\theta_b) - m_b l_b \dot{\theta}_b^2 \sin(\theta_b) - d \tag{65}$$

For the wheel part of equation (30), we can add the disturbance component d multiplied by the corresponding wheel radius l_w. Thus resulting in an equation of

(66) [49, Section 3.6].

$$\left(\frac{I_w}{l_w}+l_w m_b+l_w m_w\right)\ddot{x}_w = -m_b l_b l_w \ddot{\theta}_b \cos(\theta_b)+m_b l_b l_w \dot{\theta}_b^2 \sin(\theta_b)-T_f+T_m+l_w d \quad (66)$$

When equations (64) and (66) are combined, the disturbance modified unlinearized EOM's can be seen in equation (67) [49, Section 3.6].

$$\begin{cases} (I_b+m_b l_b^2)\ddot{\theta}_b \\ = +m_b l_b g \sin(\theta_b) - m_b l_b \ddot{x}_w \cos(\theta_b) - \frac{K_t}{R_m} v_m + \left(\frac{K_e K_t}{R_m}+b_f\right)\left(\frac{\dot{x}_w}{l_w}-\dot{\theta}_b\right) + l_b \cos(\theta_b) d \\ \\ \left(\frac{I_w}{l_w}+l_w m_b+l_w m_w\right)\ddot{x}_w = \\ -m_b l_b l_w \ddot{\theta}_b \cos(\theta_b) + m_b l_b l_w \dot{\theta}_b^2 \sin(\theta_b) + \frac{K_t}{R_m} v_m - \left(\frac{K_e K_t}{R_m}+b_f\right)\left(\frac{\dot{x}_w}{l_w}-\dot{\theta}_b\right) + l_w d \end{cases} \quad (67)$$

When equation (67) is linearized just like in the equations of (31) where the nonlinear terms were eliminated at the equilibrium by setting their values equal to zero, the resulting state-space representation is then derived into equation (68). The linearized state-space representation of the EOM's with the disturbance component of (68) is exactly the same representation as in the State-Space of (47), but with the addition of the disturbance term d in the form of the terms b_{22} and b_{42}. The b_{21} and b_{41} terms correspond to the motor input voltage v_m and the b_{22} and b_{42} terms correspond to the disturbance component d [49, Section 3.6].

$$\begin{cases} \begin{bmatrix} \dot{x}_w \\ \ddot{x}_w \\ \dot{\theta}_b \\ \ddot{\theta}_b \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & 0 & 1 \\ 0 & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_w \\ \dot{x}_w \\ \theta_b \\ \dot{\theta}_b \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ b_{21} & b_{22} \\ 0 & 0 \\ b_{41} & b_{42} \end{bmatrix} \begin{bmatrix} u \\ d \end{bmatrix} \\ \\ y = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_w \\ \dot{x}_w \\ \theta_b \\ \dot{\theta}_b \end{bmatrix} \end{cases} \quad (68)$$

The resulting equation of (69) is exactly the same as the corresponding equation of (45), except for the term b_{21} would be b_{22} and b_{41} would be b_{42}. The same resemblance is found with the term β_{11} that would be β_{12} and β_{21} that would be β_{22} in (46) [49, Section 3.6].

$$\begin{bmatrix} b_{22} \\ b_{42} \end{bmatrix} := \frac{1}{\gamma_{11}\gamma_{22}-\gamma_{12}\gamma_{21}} \begin{bmatrix} \gamma_{22} & -\gamma_{12} \\ -\gamma_{21} & \gamma_{11} \end{bmatrix} \begin{bmatrix} \beta_{12} \\ \beta_{22} \end{bmatrix} = \begin{bmatrix} \frac{\gamma_{22}\beta_{12}-\gamma_{12}\beta_{22}}{\delta} \\ \frac{-\gamma_{21}\beta_{12}+\gamma_{11}\beta_{22}}{\delta} \end{bmatrix} \quad (69)$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -435.0 & -6.1 & 9.1 \\ 0 & 0 & 0 & 1 \\ 0 & 1903.4 & 62.0 & -40.0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 \\ 20.5759 & 2.1054 \\ 0 & 0 \\ -90.0275 & 2.0256 \end{bmatrix} \quad (70)$$

In the matrix (70), the resulting numerical definitions of the calculated values are exactly the same as in the matrix (48), except for the added disturbance values of d in the second column of matrix B for the b_{22} and b_{42} part with the zeros at b_{12} and b_{32}. As already mentioned, the two different columns of matrix B are clearly not linearly dependent of each other, but instead are in correspondence with the reference values [46] [49, Section 3.6].

3.2.3 Converting the PID to Discrete Domain

Before the PID controller can be implemented into the real robot, its transfer function $C(s)$ has to be discretized, because the controller is of digital type [49, Section 3.8]. The analog (or continuous) PID controller was defined into the s-plane in equation (54). As the Arduino is a digital controller board [11], it takes an analog input signal and converts it into a digital signal and back to analog again and feeds it to the output with signal converters. For discretization, we will use the Zero-Order Hold (ZOH) [17, Section 8.3.2] [52] method because this method reconstructs the signal with the help of the built-in Digital-Analog Converter in the Arduino. In order for the discretization to succeed, the sampling time must be chosen correctly. If too large a sampling interval [17, Section 8.3.6] is chosen for the Analog-Digital Conversion, it could make the robot system unstable [11].

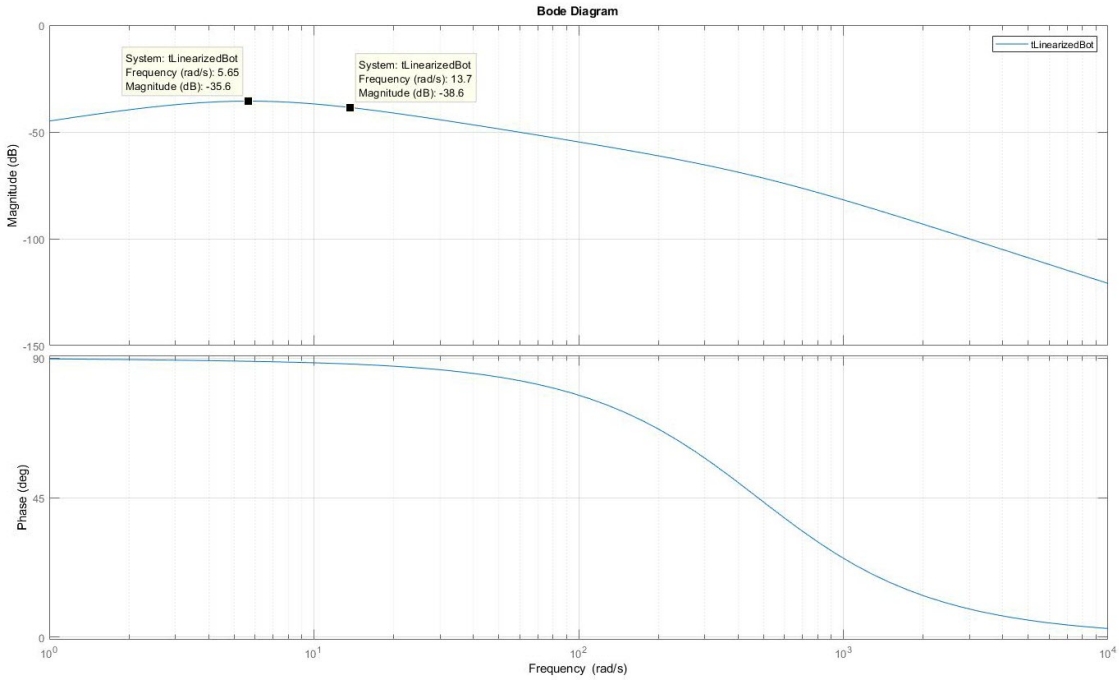

Figure 12: The Bode plot of the PID controller TF [5].

From Figure 12, the bandwidth is set to the frequency where three desibels (dB) are lost from the peak magnitude of the graph. The Data Cursor in the Bode Diagram gives about 5.65 rad/s frequency at peak magnitude and 13.7 rad/s around -3 dB. As

$$1 \quad \frac{rad}{s} = \frac{1}{2 \times \pi} \quad Hertz,$$

then 13.7 rad/s is about 2.2 Hertz. According to the sampling theorem, a system should be sampled at a rate that no aliasing occurs. Aliasing is a result of undersampling, while oversampling has no effect on the reconstructed outcome [30]. But too frequent a sampling rate could cause a loss of accuracy. When the sampling rate is lowered, the computer has more time for calculations, thus usually the best sampling rate is the slowest possible, while keeping the performance-cost-ratio at a decent level. But when converting a system to a discrete domain, there is a fictious limit that is inherited from the approximation in the conversion, which might decrease performance or even cause instability if the sample rate is lowered too much [17, Section 8.5]. The sampling period T is given by the sampling frequency and is used in the transformation of the TF to the discrete domain [35]. If the robot's output signal is sampled at 110 Hertz, it would mean that the frequency interval would become 4.4 Hertz. The sampling frequency is taken 25 times the cutoff frequency. Thus the sampling period is

$$T = \frac{1}{110} \approx 0.00909 \approx 0.01.$$

Simulink doesn't allow other than fixed step sizes for the sampling time for the simulation model `LabA_LinearizedBotVsSimulator_Discrete.slx`, thus the sampling period is rounded up to a 0.01 [49, Section 3.8].

$$\begin{aligned} C(z) &= (1 - z^{-1})\left(\mathcal{Z}\left(\frac{k_P}{s}\right) + \mathcal{Z}\left(\frac{k_I}{s^2}\right) + \mathcal{Z}(k_D)\right) \\ &= \left(\frac{z-1}{z}\right)\left(k_P \frac{z}{z-1} + k_I \frac{Tz}{(z-1)^2} + k_D\right) \\ &= k_P + k_I \frac{T}{(z-1)} + k_D \frac{z-1}{z}. \end{aligned} \tag{71}$$

In the calculations of (71), the continuous-time controller from equation (54) is discretized using the z-transform table [51]. To get the z-transform, the corresponding terms are multiplied by $\frac{1}{s}$ and $z - 1$ for convenience, z-transformed and divided by z before the next step. ZOH holds the previous value of a zero-order polynomial and changes the value in the next sample time, which results in a series of small step functions. The T represents the sample time of the discrete system and the s or z the complex variable, which relate to each other, from the Laplace-transform. The z-transform is the summation of all of the positive integers k and it converts the time domain signal into z domain signal. The ZOH samples the output in the specified sample time T to the discreet output $Y(z)$. The input signal u has a value every T seconds in k index. [13] With the aforementioned preparations, then by the z-transform table, the unity with regards to k_P of equation (54) equals to

$$1 = \frac{z-1}{z} \times \frac{z}{z-1},$$

the $\frac{1}{s}$ with regards to k_I equals to

$$\frac{1}{s} = \frac{z-1}{z} \times \frac{Tz}{(z-1)^2}$$

and the s with regards to k_D equals to

$$s = \frac{z-1}{z} \times 1.$$

Then after cleaning up the equation, the final form of the discretized PID controller transfer function in (71) is reached [49, Section 3.8].

3.3 Simulating the PID Controlled Robot Behavior

To make sure that the robot behaves as expected, one should make an experiment on it by making the Simulink simulate a small poke on the linearized model of the robot (equation (70)) when there are only the essential parameters input into the system. The reason why a small poke is simulated is that a forceful poke would be almost impossible to stabilize without making hardware changes to the actual robot and because the stable region for pokes to be applied was set to be near. For now, we will work on continuous-time for the first experiment before the implementation to the real robot, because we do not need the discrete equivalent to be uploaded into the Arduino yet. The inputs will then be the motor input voltage v_m and the disturbance d from the B matrix and the outputs will be all the states (θ_b and x_w). The two matrices, C and D must be updated to the ones on (72). [49, Section 3.7]

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{72}$$

For the disturbance signal, the small values of 0.1 and zero of (73) should be used [49, Section 3.7].

$$d(t) = \begin{cases} 0.1 & for \quad t \in [0, 0.01] \\ 0 & otherwise \end{cases} \tag{73}$$

With these implemented parameters, we are now interested in checking the output results for the body angle θ_b and the wheel position x_w as determined already in equation (3). We need to create the disturbance signal $d(t)$ with the Simulink's Signal Builder in `LabA_LinearizedBot.slx`. The PID Controller block, which is the way to implement the PID in Simulink, requires to set its filter pole location determining Filter coefficient (N) [15] to a suitable value or leave it to the default of 100 by double clicking the corresponding block in Simulink to open up the Block Parameters. If the Filter coefficient is changed, MATLAB might have to be restarted before another change in the coefficient could be visualized. The Filter coefficient doesn't change the output of the system if it is changed further positive or a small positive value and it isn't allowed to be changed into a negative value because the Simulink gives an error related to time finiteness. Even if the Filter coefficient value is set to a very small value, the output of motor input voltage doesn't change. Only when the Filter coefficient is set to very large, the input motor voltage changes to 1.578 Volts, which means the simulated poke would be a bit more forceful and the

wheels would turn a little further. The default value was used for the (N) in creating the graphs, and it should numerically be determined by the values of the k_D in (60) [49, Section 3.8]. This is proved by setting the Filter coefficient Compensator formula equal to -0.0969, which is the value of k_D in (60) and getting the same value as a result when $(N = 100)$ is used like this

$$-0.0969 = \frac{100}{1 + \frac{100}{s}} \quad <=> \quad -0.0969 = \frac{100s}{s + 100}$$
$$<=> \quad -0.0969(s + 100) = 100s \quad <=> \quad -0.0969s - 100s = 9.69$$
$$<=> \quad -100.0969s = 9.69 \quad <=> \quad s = -0.0969.$$

This would set the motor input voltage to 1.164 Volts, which is a convenient amount of input voltage close to the maximum output voltage (1.2 V) of one of the batteries used in the robot assembly and is in a small one volt region. The graphs of Figures 14 and 15 can be obtained after running `LabA_Solutions_LoadPhysicalParameters.m`, `LabA_Solutions_LoadStateSpaceMatrices.m`, then `LabA_Solutions_ComputePIDGains.m` and finally the file for the transfer function `LabA_Solutions_ComputeTransferFunction.m` in MATLAB in this order and then `LabA_LinearizedBot.slx` in Simulink and after that once more in MATLAB `LabA_Solutions_LoadPhysicalParameters.m` from [46]. [49, Section 3.7]

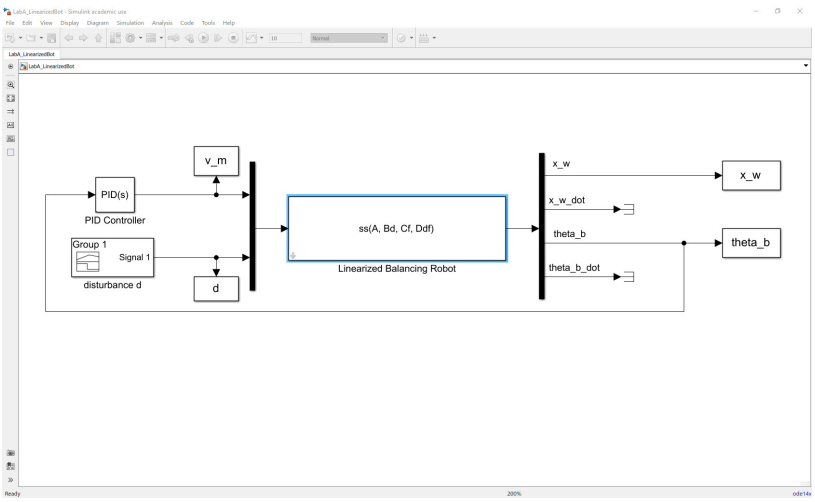

Figure 13: The Simulink model of the linearized system [49, Section 3.7].

In Figure 13, the Simulink block diagram of the robot is presented. When the linearized system with the updated matrices C and D parameters are run in MATLAB, the resulting graphs of the opportune functions are drawn in Figure 14 [49, Section 3.7].

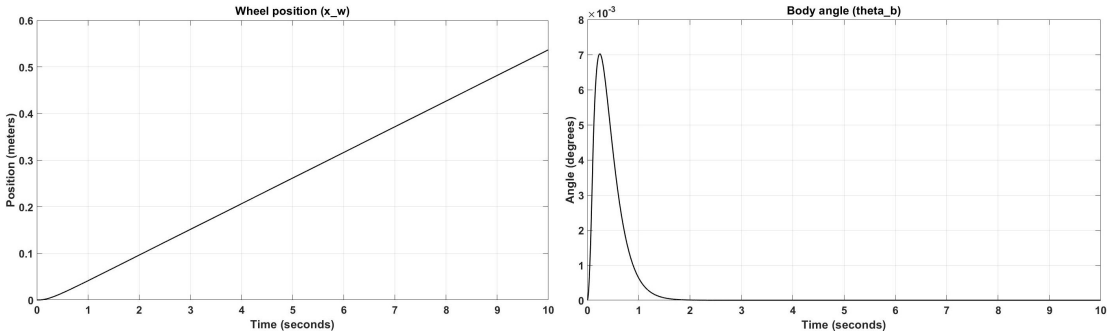

Figure 14: The wheel position and body angle [49, Section 3.7].

As can be seen from Figure 14 that the robot system integrates the wheel position x_w, which means that when the robot is poked, it starts moving. From the body angle graph, it can be noted that after the poke, the PID controller starts to regulate θ_b. The controller doesn't care of the position of the wheel and thus the velocity of the wheel either, so the robot will start to wander off from the initial position uncontrollably. This means that the system is still unstable. There is only one controller that cannot control both the wheel position and body angle, so eventually it will need another PID controller for the wheel position as a counterpart. The wheel position controller can tell the wheel position from a rotary encoder, which is embedded inside the Lego motor. [49, Section 3.7]

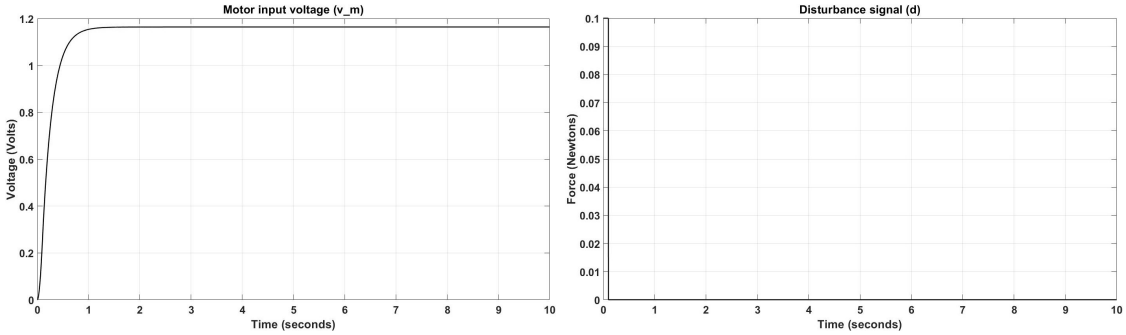

Figure 15: The motor input voltage and disturbance [49, Section 3.7].

In Figure 15, we can see the motor input voltage and the dirac delta [12] shaped disturbance signal that is caused by the simulated robot poking effect [49, Section 3.7].

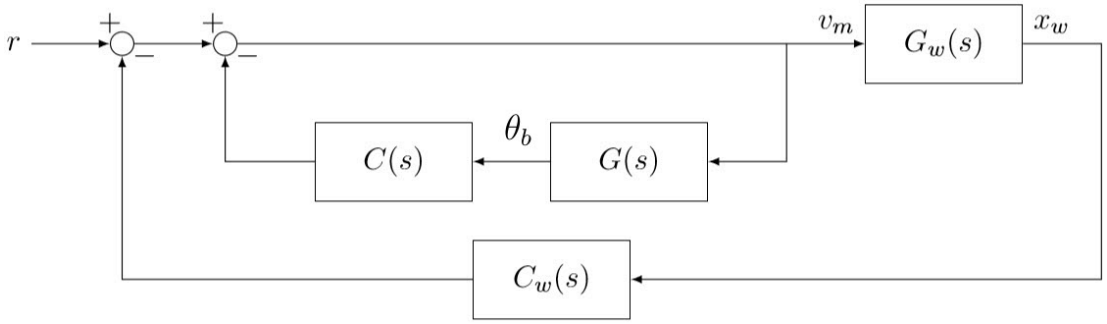

Figure 16: The Double-PID block representation [49, Section 3.7].

Figure 16 shows the cascading PID controller $C_w(s)$ that controls the wheel position from v_m to x_w. The other blocks are derived from equation (54) for the PID controller $C(s)$ for the body angle and the transfer function $G(s)$ (55) from v_m to θ_b. The cascading controller is not very simple to design, as $C_w(s)$ affects $G(s)$ and $C(s)$ affects $G_w(s)$. The problem is of MIMO type (not SISO for single input and single output) because of two inputs (v_m and d) and two outputs (θ_b and x_w) and can be solved with state-space formulations, but that design process is saved for later and now we will just do the controller for the body angle θ_b. [49, Section 3.7] Next thing to do is to simulate the robot controller prototype on a visual robot simulator, because this is easier than to do it in a real robot [49, Section 3.9].

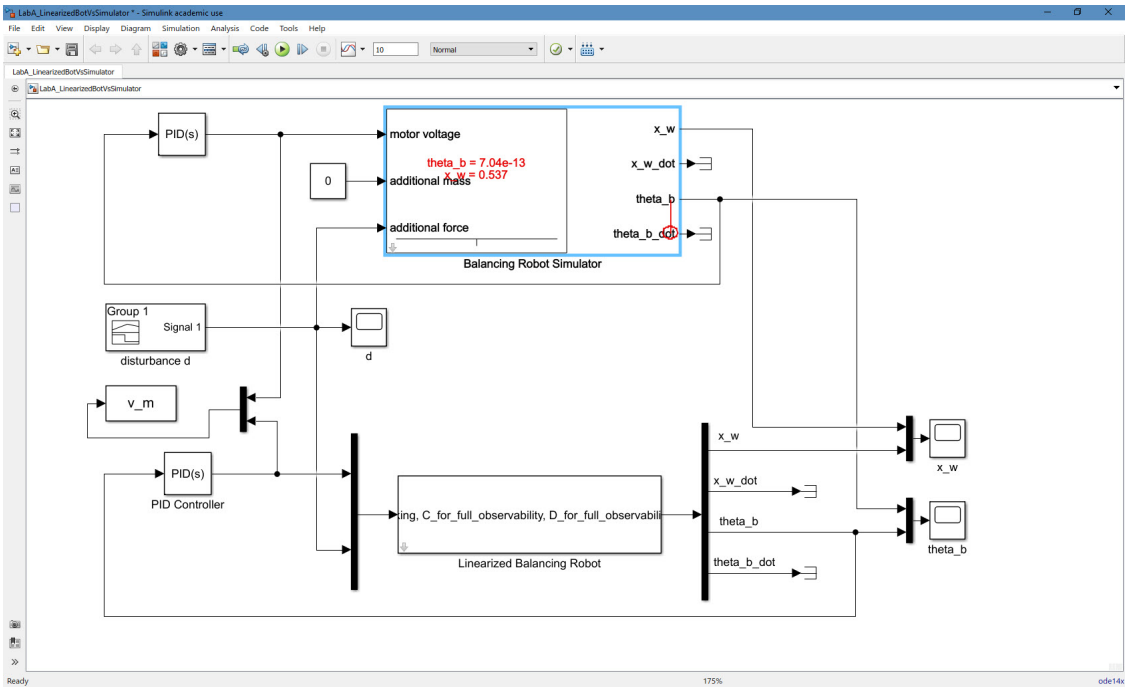

Figure 17: The robot prototype simulator [49, Section 3.9].

For the second and last experiment before trying the PID controller in reality, in Figure 17, the robot functions are prototyped by running the Simulink file of `LabA_LinearizedBotVsSimulator.slx`. The red circle with a pole represents the robot and one can notice that after the simulator was run, the robot moved 53.7 centimeters forward with a small 0.01 disturbance signal and within 10 seconds of simulation time. There is no harm in playing with the simulator, as nothing can be broken. When the simulator is double-clicked, one could adjust its parameters, but in this case, the parameters are left at their default values. This simulator implements the nonlinear EOM's that were derived in equations of (67) and approximates them with enough good an accuracy. With the simulator, one can change the mass of the body and the disturbance level. The simulator also shows the whole states of the system i.e. the body angle velocity $\dot{\theta}_b$, the body angle θ_b, the wheel velocity $\dot{x}_w$ and the wheel position x_w. With the simulator, we don't yet need any state observers to implement the full state controllers, but they will be tested and characterized later in this Thesis for their performance. There are three things to check from the

simulator: If the controller is stabilizing the nonlinear dynamics, if the linearized dynamics approximate the nonlinear dynamics as expected and if the chosen sampling is time good for control? We would now need to to implement two Simulink schemes. One for control with the continuous-time PID $C(s)$ with the disturbance d and one for control with the discrete-time PID $C(z)$ with disturbance d. For both of these schemes, the value of $\overline{d}$, which makes the robot fall after the simulation is started from the equilibrium, should be determined for the disturbance d seen in equation (74). [49, Section 3.9]

$$d(t) = \begin{cases} \overline{d} & for \quad t \in [0, 0.01] \\ 0 & otherwise \end{cases} \tag{74}$$

The comparison of the trajectories of both nonlinear and linearized dynamics of the terms $(\theta_b, \theta_b^{lin}, v_m, v_m^{lin})$ is obtained next. The simulator for the discrete prototype is found from `LabA_LinearizedBotVsSimulator_Discrete.slx`. For the discrete equivalent, the fSamplingPeriod inside the two PID(z) blocks in the Sampling time section is controlled from MATLAB's script `LabB_CheckCommunications_Parameters.m` and is set to the aforementioned value of 0.01. Also the Integrator method there is changed to Backward Euler from Forward Euler compared to the continuous case in order to avoid numerical problems as suggested in the reference [49, Section 3.9]. As it was found out empirically, in the continuous case: $\overline{d} = 0.78$ robot falls, $\overline{d} = 0.77$ robot doesn't fall. In the discrete case exactly the same results were obtained and the similar graphs drawn for θ_b and v_m as with $\overline{d} = 0.78$ the robot falls and with $\overline{d} = 0.77$ the robot doesn't fall. As also empirically found out, both the continuous and discrete controllers follow the simulator output in terms of body angle and wheel position side by side so the answer to those three things to check would be yes to all. The miniscule .009 motor input voltage difference between the linearized PID controller and the prototype simulator on the continuous system can be seen in Figure 18. The difference is exactly the same on the discrete case. [49, Section 3.9]

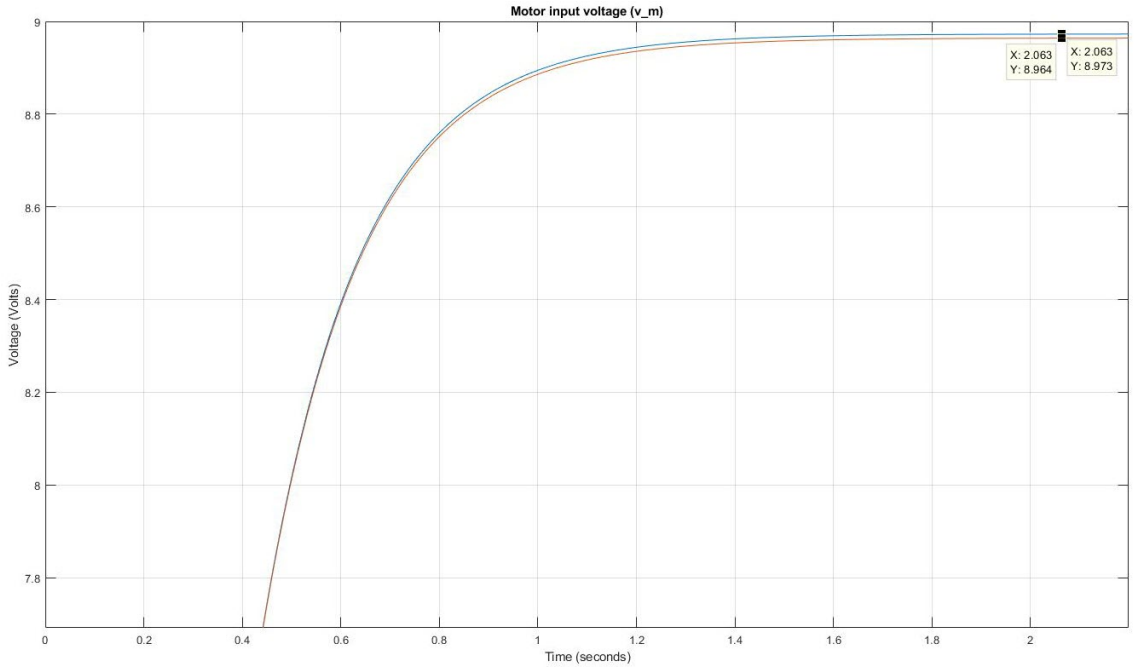

Figure 18: The difference between the robot prototype simulator and the linearized controller [49, Section 3.9].

3.4 Field-Testing the PID Controller

To test the PID controller on the real robot, we now experiment with the continuous version and then tune it and discretize its results. The two ways to run tests used here are as follows. One is to test that the robot is commanded by Simulink by pressing (Ctrl+T). The other one is that the robot is loaded with code (Ctrl+B). In both cases, the communication is terminated by the reset button located on the robot itself but also when the code is loaded into the robot, it can be terminated by the battery switch. The difference to the reset methods are that the reset button only temporarily resets the robot and then after a few of seconds establishes the communication again. If there is a problem uploading the code to the robot, the Simulink's automatically creates a temporary code folder `NameOfTheSimulinkDiagram_rtt` that should be deleted from the hard drive before the next run. [49, Section 4.1] To establish the communications with the MinSeg, one should do the following when using Windows 10 64 bit setup. If there are other MATLAB versions than R2017b or other Arduino versions than Arduino for Windows 10 app installed in the computer used for this MinSeg simulation, then uninstall those unless absolutely necessary to not uninstall them. Install Arduino Window10 app from https://www.arduino.cc/. Then install MATLAB R2017b with all packages and not only the recommended ones, just in case. Install the packages even if the RASPLib is currently recommended to be installed into a 32 bit operating system, e.g. Windows 7. Install the MATLAB Add-Ons of Arduino support for MATLAB, Arduino support for Simulink and the Rensselaer package. Download and unzip the latest RASPLib folder $hurstj01 - RASPLib - 63c29b7$ to `C:\MATLAB` and download and unzip the Code.zip from R7003E 2017 course material web page to `C:\MATLAB` if this has not been done in earlier steps. See that the

MatlabAndSimulink folder from Code.zip in R7003E 2017 course material with Rasplib and startup.m are in the `C:\MATLAB` folder. Assemble the MinSeg robot with one leg and two wheels setup, which has been the basis for the mathematics on this Thesis. Connect the MinSeg to the computer running the MATLAB program with its USB cable. Check the number of the COM port from the Control Panel in Windows 10. Open Matlab R2017b. Set the Current Folder to `C:\MATLAB\MatlabAndSimulink`. Run `LabB_CheckCommunications_Parameters.m` from that Current Folder and see that the fSamplingPeriod matrix appeared into the workspace unless it is there already. Open Simulink by typing simulink into the MATLAB console. In Simulink, open a blank model. Then open the Rensselaer package from the Library browser by left clicking it and from the Demo files, open the M2V3.2 Mega Demo file, which will pop up a few windows to the desktop. Go back to MATLAB and click the 'Run' button to run startup.m. If MATLAB prompts about add to path, then click add to path. Open the file `LabB_CheckCommunications.slx` by double clicking it from the Current Folder. See that the COM port is set to the one holding the ArduinoMega2560 from the Model Configuration Parameters in Simulink. If it is not set, then set it manually in the Hardware Implementation section. Double click the center block in `LabB_CheckCommunications.slx` to see that the block with the Arduino pins are not grey. If they are grey, see that the Arduino Windows10 app is installed into the PC. Click the 'load parameters' text in the Simulink diagram. When Simulink prompts 'parameters loaded successfully', click ok. If an error occurs, see that the Current Folder in MATLAB is set to the `C:\MATLAB\MatlabAndSimulink` and click 'load parameters' text again. In Simulink, click the green 'Run' button and wait for the Code Generation Report window to pop up. When the `LabB_CheckCommunications.slx` diagram turns red and no error occurs, click the Gyro block and move around the MinSeg robot to see a changing graph. Autoscale the graph to see the graph better. Do the same for the accelerator, motor and encoder blocks. If Simulink gives an error message about not finding the SupportPackages, uninstall and install the previously mentioned Add-Ons, remove the contents of the `C:\MATLAB` folder and unzip the RASPLib folder and Code.zip again, unplug the MinSeg USB cable from the PC, close MATLAB and Simulink and restart the computer and try again. If you see the graphs drawing curves online when you click on the gyroscope, accelerometer, motor and encoder blocks (i.e. scopes), then everything works. Stop the simulation by clicking the 'Stop' button in Simulink. Now that the communication link is established with the robot, other windows, such as the Demo windows can be closed and the simulation can be started again. But before that, if one desires to plot graphs and save them, then the startup.m file should be ran and 'add to path' clicked. Then the image of the drawn graph can be saved. To re-establish the connection, press the reset button on the MinSeg robot and click 'load parameters' text and click 'Run' again. No hardware or software changes after the connection to the robot works should be made. Also everything should be opened in R2017b and not under other MATLAB versions. If there are problems, one should make sure the MinSeg robot is connected to the PC running the MATLAB version R2017b and see that everything always opens in R2017b and not in other MATLAB versions. To debug, first it should be made sure that the

battery switch is ON in the robot. In MATLAB, one should select Up One Level until the MatlabAndSimulink folder is reached. Then cd is typed in the MATLAB console to see if it is the Current Folder. Then simulink is typed into the MATLAB console and the file `LabB_CheckCommunications.slx` should be opened. After this one should do the following steps. Type arduino into the MATLAB console and click the Add-on explorer Support Package installer link and install the MATLAB Arduino support package. To debug in Simulink, open `LabB_PIDOverRobot.slx`, check the COM port from Hardware Implementation - Target Hardware resources - Host-board connection and run `LabB_TuneTheGyro_SaveParameters.m` to create the `GyroBias.mat` for the next step of the file `LabB_PIDOverRobot_Parameters.m`. Then press (Ctrl+B). Also, to make sure, press (Ctrl+E) to see that there is the Arduino Mega2560 text visible in Hardware board section of Hardware Implementation and if not, click Get Hardware Support Packages. Then go to the Device details at the Hardware board section, click Host-board connection and select Set host COM port: Manually and Set COM port number. Then click Apply and OK and save the file. [49, Section 4.2] The accelerometer and gyro behaviors are not ideal in real life. Their measurement signals are noisy, so they need bias tuning, which can be done in the file `LabB_TuneTheGyro_SaveParameters.m` and during the tuning, the COM port of file `LabB_TuneTheGyro.slx` might need to be fixed also. During the code execution, the robot should be kept standing still and upwards as it should be like when it has balanced itself perfectly. This code execution procedure lasts about 30 seconds, where the robot is taught how to stand when it balances itself. The resulting variable fGyroBias is saved in the file `GyroBias.mat`. This calibration step is crucial for obtaining the expected operation of the MinSeg. [49, Section 4.3] The batteries had a 75 percent charge before they were secured to the robot's battery case, but the less then full charge of the batteries didn't seem to be a problem as the robot worked in a perfectly satisfactory way. Before clicking the 'load parameters' text in Simulink, the connection to the robot was established. In short, the sequence to connect to the robot is:

1. Switch to robot battery mode (BATT ON).
2. Connect USB.
3. Open MATLAB `LabB_CheckCommunications_Parameters.m`.
4. Set Current Folder to `C:\MATLAB\MatlabAndSimulink`.
5. Run `LabB_CheckCommunications_Parameters.m`.
6. Open Simulink by typing simulink in the MATLAB console.
7. Open Rensselaer Arduino Support Package from a Blank Model Library Browser.
8. Open Demo M2V3.2 Mega (Right click Open M2V3.2 Mega library).
9. Run startup.m from `C:\MATLAB` and add it to the path.

10. Open `LabB_CheckCommunications.slx`.
11. Correct COM port from Model Configuration Parameters.
12. Click 'load parameters'.
13. Run `LabB_CheckCommunications.slx` and balance the robot in vertical position for 30 seconds.
14. Stop the simulation, switch the battery robot mode to (5 V USB) and disconnect the robot from the PC.
15. Close other windows than MATLAB and `LabB_CheckCommunications.slx`.

When calibration of the robot is done successfully, the fGyroBias variable in `GyroBias.mat` value will change and in this case it changed from -230 to -238 after calibration. After the calibration, the file `LabB_TuneTheGyro_SaveParameters.m` should be ran again in MATLAB (which then changed its value from null to 0.005), 'load parameters' pressed from the file `LabB_PIDOverRobot.slx` and then Deploy to Hardware (Ctrl+B) pressed in Simulink to tune the gyro and make the robot balance itself better. In the calibration step after pressing 'Run', the robot is held in its perfectly balanced position until the Code Generation Report window pops up and MATLAB is not busy anymore. Two heavy boxes were used to support the robot from both sides so that it didn't need to be held with both hands while doing the calibration, which otherwise proved to be tricky. After a few calibration practice runs, the boxes were not needed anymore. Then when Simulink is in 'Ready' state and the robot motor turns on, which is hearable, the USB-cable from the robot should be disconnected and the robot placed upright in a spacy, level and horizontal platform after which the robot power switch should be turned to battery and lastly the robot released. When the experiment is wished to end, the power should be switched to USB in the robot, so that it would stop moving. When one wishes to save graphs from the robot experiment, then the robot should be placed upright again, power switched to battery and the file `GetDataFromSerial.m` ran in MATLAB. After this, when the file `PlotDataFromSerial.m` is ran, the graphs can be saved. The experiment showed a fault in the motor port M1, because it didn't work, but M2 port did and was used in the field-tests. The M1 motor port is probably inoperable and broken in some way as it could not be fixed. After about 10 minutes of testing, the robot was able to balance itself for one second with this continuous PID controller before falling on its face or on its back. The impact to the surface was prevented with hands, so no harm was done for the robot itself. Figure 19 shows a result of the gyroscope measurements of the first successful test run on the robot. The test was commenced while the robot was flat on its back on the surface and then lifted up and turned around between the 15 and 20 second mark in the approximately 37 second long test run. The downward slope in Figure 19 tells that there is a drift from the measurements and is probably caused by a calibration error in the sensor. [49, Section 4.4]

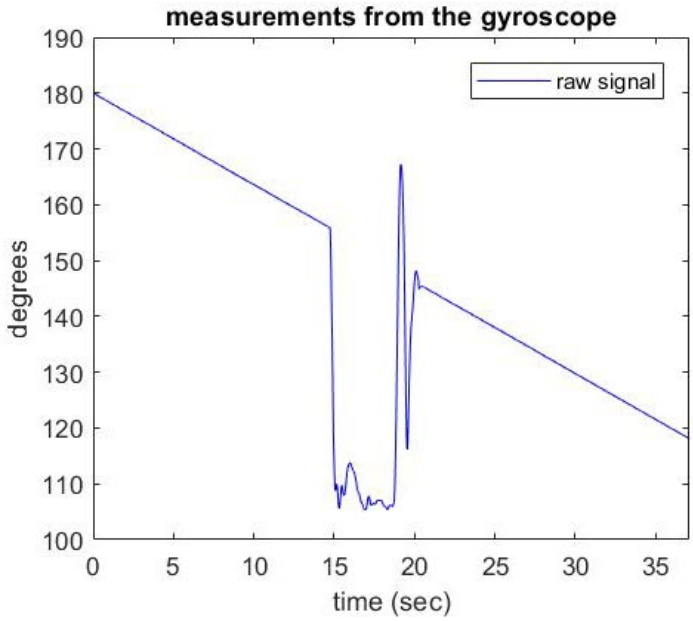

Figure 19: The measurements of the gyroscope of the body angle θ_b.

Another test showed the accelerometer signal for a 30 second period in Figure 20. The wheel position x_w changed in about 10 centimeter range back and forth after the MinSeg was released from the equilibrium position while the falling was prevented with hand support [49, Section 4.4].

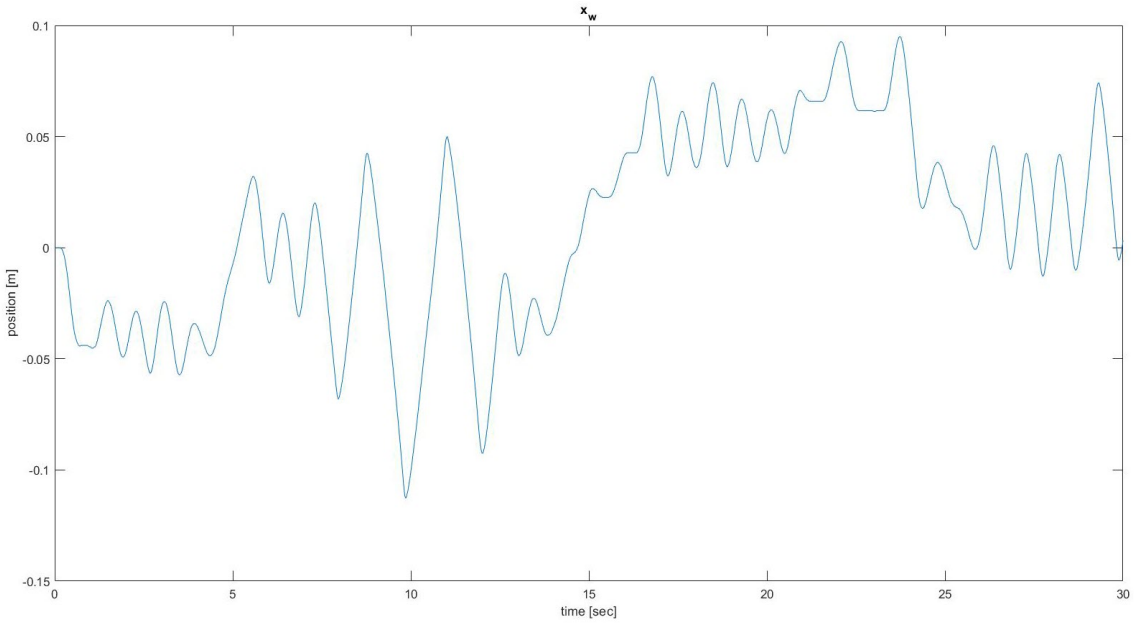

Figure 20: A 30 second accelerometer test result.

4 Designing an Improved PID Controller

While the field-tests showed that the robot cannot be stabilized with the current PID controller, there is a solution to improve the PID design. To this aim, first the controllability and observability of the system from (32) need to be calculated. Based on the findings in (76), which is calculated based on the matrix (47), the controller doesn't care about the whereabouts of the robot wheel x_w, thus there is no observability, which in turn does cancellations into the transfer function. After the controllability and observability matrices are calculated, there is a need to design two State-Space controllers, where one uses poles allocation and one the LQR method. But to design the State-Space controller for either method, we must pretend that we have the wheel position information for the full state of the system and test the controllers in simulations [49, Section 4.6]. After that the state observer can be designed and added to the simulator. After this, both the controller and observer are also added to the simulator since they are discretized and finally when the simulations show that in theory everything works, this control strategy is put to the field-test anew. The valuable wheel position information needed for the improvements to be implemented is recovered by the robot motor's encoder. [49, Section 4.5]

4.1 Introduction

For the controllability matrix, the results are displayed in the matrix (75) [49, Section 4.5].

$$C = \begin{bmatrix} B & AB & A^2B & A^3B \end{bmatrix} = \begin{bmatrix} 0 & 20.58 & -9773.71 & 4643126.0 \\ 20.58 & -9773.71 & 4643126.02 & -2205805265.8 \\ 0 & -90.03 & 42763.68 & -20318610.2 \\ -90.03 & 42763.68 & -20318610.22 & 9652741765.7 \end{bmatrix} \tag{75}$$

In (75), the results show full controllability. For the observability matrix, the results are displayed in the matrix (76) [49, Section 4.5].

$$\mathcal{O} = \begin{bmatrix} C & CA & CA^2 & CA^3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1.00 & 0 \\ 0 & 0 & 0 & 1.0 \\ 0 & 1903.44 & 62.02 & -39.97 \\ 0 & -904146.27 & -14088.27 & 19049.09 \end{bmatrix} \tag{76}$$

Where the results show only partial observability due to the aforementioned absence of the position information of the wheel x_w [49, Section 4.5]. As the state description, as seen in both matrices (75) and (76) is nonsingular (i.e. negative) [38], we cannot use the control canonical form, which would make the state feedback gains easy to design, thus the control design is as follows [17, Section 7.4]. The block diagram of the system is displayed in Figure 21, where one can see that the wheel position x_w is not observable as it does not output y. The output does not get out, because until now, there has been no interest in the wheel position observation. In the matrix (47), it can be noticed that the system is not controllable for the wheel velocity state

$\dot{x}_w$, because the value for the term of b_{11} in the B-matrix is zero. Thus there is no input signal u to control the wheel velocity. Besides that there is no wheel position information gathered from the system matrix A because the term a_{11} is zero. In fact, no wheel position information is gathered from any of the suitable states because of the terms a_{21}, a_{31} and a_{41} also being zero in the A matrix. Nor is the system observable for the wheel velocity state $\dot{x}_w$ because of the wheel position term x_w being zero in the output matrix C for the output y in (47). The wheel acceleration on the second row of the A matrix is controllable because of the existence of the b_{21} term in the controllability matrix B and not observable because of the wheel velocity term $\dot{x}_w$ being zero in the output matrix. For the body angle velocity state $\dot{\theta}_b$, the system is not controllable, because there is zero in the b_{31} term of matrix B, but on the other hand it is observable, because of the term θ_b in the output matrix, which can also be seen in Figure 21. The body angle acceleration $\ddot{\theta}_b$ is controllable because of the term b_{41} in the output matrix, but this state also is not observable because there is no output for the body angle θ_b in the C matrix. [17, Section 7.4]

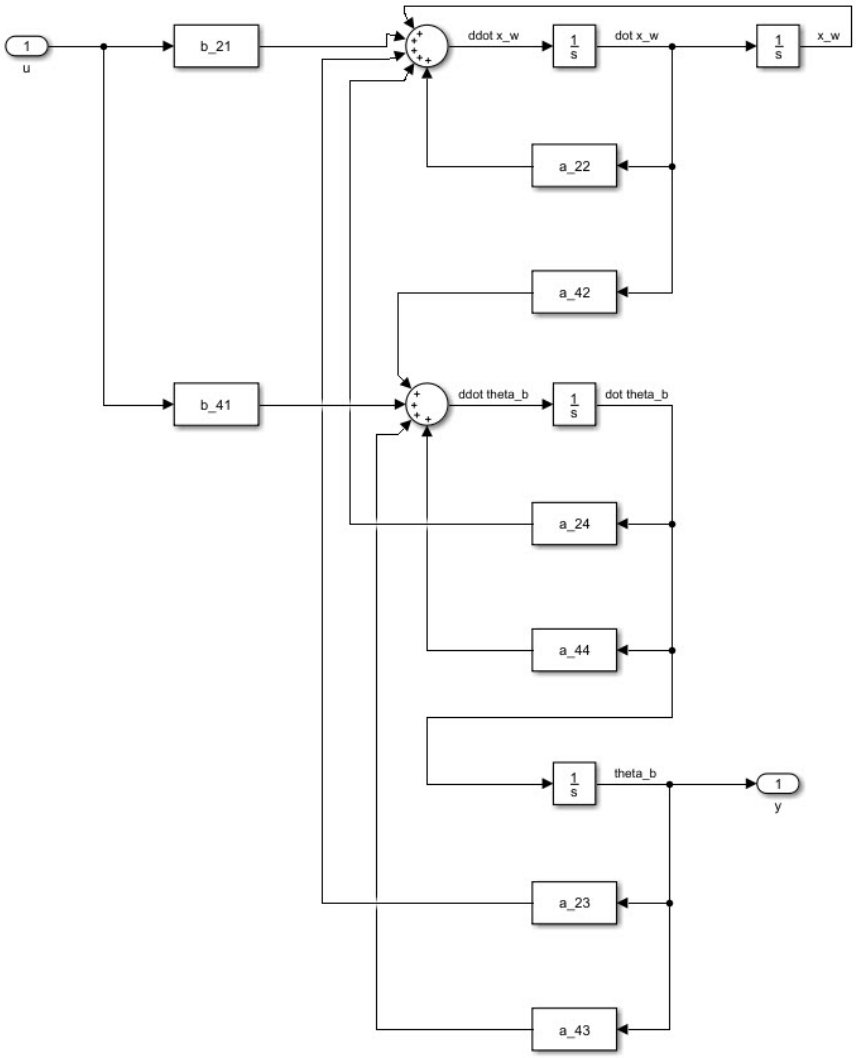

Figure 21: The block diagram of the MinSeg.

4.2 Improving the Simulator

4.2.1 Designing a State-Space Controller

So now that we know the controllability and observability of the balancing robot system, the construction of the State-Space controller by second order approximated pole selection can begin. To select the location of the poles, one needs to determine a proper second order system. For gains matrix K, the values $[-3, -475, -3, -3]$ were used and they gave as K values $[-17.6405 - 38.8205 - 52.5060 - 8.9724]$, when the Ackermann's formula [2] was used in MATLAB (MATLAB command: acker) because we have full controllability (75). Ackermann's formula is useful when systems are not easy to express in the control canonical form. By running `LabB_ControllerOverSimulator_Continuous_Parameters.m` for the values of K and `LabB_ControllerOverSimulator_Continuous.slx` for the simulations, the corresponding graphs for the motor input voltage v_m and body angle θ_b were obtained into Figure 22. [49, Section 4.6]

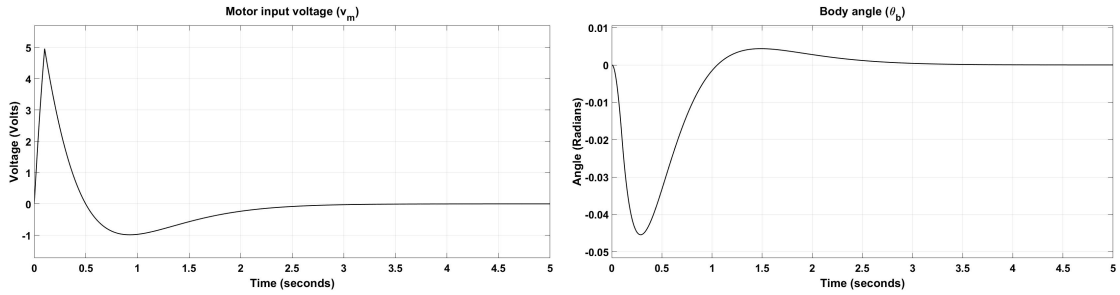

Figure 22: The motor input voltage and robot body angle [49, Section 4.6].

Figure 22 shows the system response to the robot being released from the equilibrium. The body starts to move, so the motor voltage controlling the body movement starts to rise. When the motor voltage has reached high enough level, the body begins to move back towards the equilibrium point and reaches it finally. At that time, the motor voltage has also lowered down to zero as the body is no longer out of the equilibrium point. Because the pole location procedure is not exact science [17, Section 7.5.1], the poles were selected as close to the slowest pole (the dominant pole) [48, Section 3.6] as possible, to the location of -3 from the origin of the stabilized transfer function (61) that should be the best starting point in determining the proper (equal number of poles and zeros or more poles than zeros) dominant second order system. The fastest pole of -475.1 was moved to -475 for calculating convenience and others moved into -3. When running these parameters, MATLAB console gives an error about the fast -475 pole that this pole location is more than 10 percent in error, but as we would not like to change system characteristics too different from the original system of (61), this error is ignored. To recap what the moved poles were before in the unstabilized transfer function, one pole was moved to -3 from $+5.7$ and another one from the origin and also one pole from -5.657, which acted as the integrator, was moved to the same spot on the left half-plane. It should be a good idea not to move the poles too much in order to avoid actuator saturation and

risk stability [49, Section 4.6].

$$\mathcal{J} = \int_0^{+\infty} \left(\rho \begin{bmatrix} x_w & \dot{x}_w & \theta_b & \dot{\theta}_b \end{bmatrix} W \begin{bmatrix} x_w \\ \dot{x}_w \\ \theta_b \\ \dot{\theta}_b \end{bmatrix} + u^2 \right) \mathrm{d}t. \tag{77}$$

For the LQR method, we first need to design a cost function J as seen in (77), for computing the gain K, which is integrated from zero to infinity to work all the time, not just a certain amount of time, because we don't really know exactly how long the robot is going to be balancing itself when the balancing is started. Inside the cost function there are the suitable states of the system: Wheel position x_w, wheel velocity $\dot{x}_w$, body angle θ_b and body angle velocity $\dot{\theta}_b$, which are added to the input vector u of J and tried to derive close to zero (a positive definite matrix [33]) as time goes to infinity, and not zero even as the equilibrium is at zero for all the states. Thus all the designed initial state values have to be positive and not zero or negative. This is because if the controller gets zero as input for any of the states in the cost function, then it can decide those zero values to become infinite by turning up the input to any high level, which is not desirable. If the controller is not penalizing any of the states, the system might explode. So we would like to have as small a numbers as possible for the states to begin with. This will be done by evaluating the initial values of the states for their maximum possible values of their given range and then weighing the states down in relation to their penalizing counterpart states to as small as possible. [17, Section 7.6.2] [25] [49, Section 4.7]

$$W = \begin{bmatrix} 1 & & & \\ & 3 & & \\ & & 10 & \\ & & & 10 \end{bmatrix} \tag{78}$$

The final values were placed in the weighing matrix W diagonally as in (78) in order for it to symmetrically take in all the corresponding states as its values where the term ρ from (77) acts as gain for the cost function. The bigger values in the weighing matrix are penalized more, so we would like to penalize the body angle and body angle velocity more than wheel position and wheel velocity, which is because we would not like the body angle to deviate too much or too fast from the equilibrium as the robot is moving in order for the robot not to fall. Thus the body states have bigger values than the wheel states [25]. The cost function is calculated by transforming the State-Space matrices A (70), B (70), C (75), and D to a transfer function and finding out its zeros and poles from the computation of both positive TF $G(s)$ and negative TF $G(-s)$ and computing their roots and gain matrix K for the controller [49, Section 4.7]. The matrix D is given a value of 1, because we would like to have the feedforward element in the system now to predict the robot behavior. After the cost function J is ran in MATLAB for the first time, one can start tweaking its state values to most suitable ones that give the commonly desired fast system response times and low-cost controllability. [17, Section 7.6.2]

[25] [49, Section 4.7] When the desired appropriate weights W are input in MATLAB file `LabB_ControllerOverSimulator_Discrete_Parameters.m`, the corresponding positive TF $G(s)$ gives

$$\frac{-838.5s^3 - 879.7s^2 - 2181s - 727}{s^4 + 475s^3 - 62.02s^2 - 1.537e04s},$$

$G(-s)$ gives

$$\frac{838.5s^3 - 879.7s^2 + 2181s - 727}{s^4 - 475s^3 - 62.02s^2 + 1.537e04s},$$

gains matrix gives out

$$K = [-0.3162 - 42.4163 - 63.0683 - 10.5755]$$

and the root locus is shown in Figure 23. The weight matrix W was calculated from the output matrix C by multiplying it with its own transpose C^T. [49, Section 4.7]

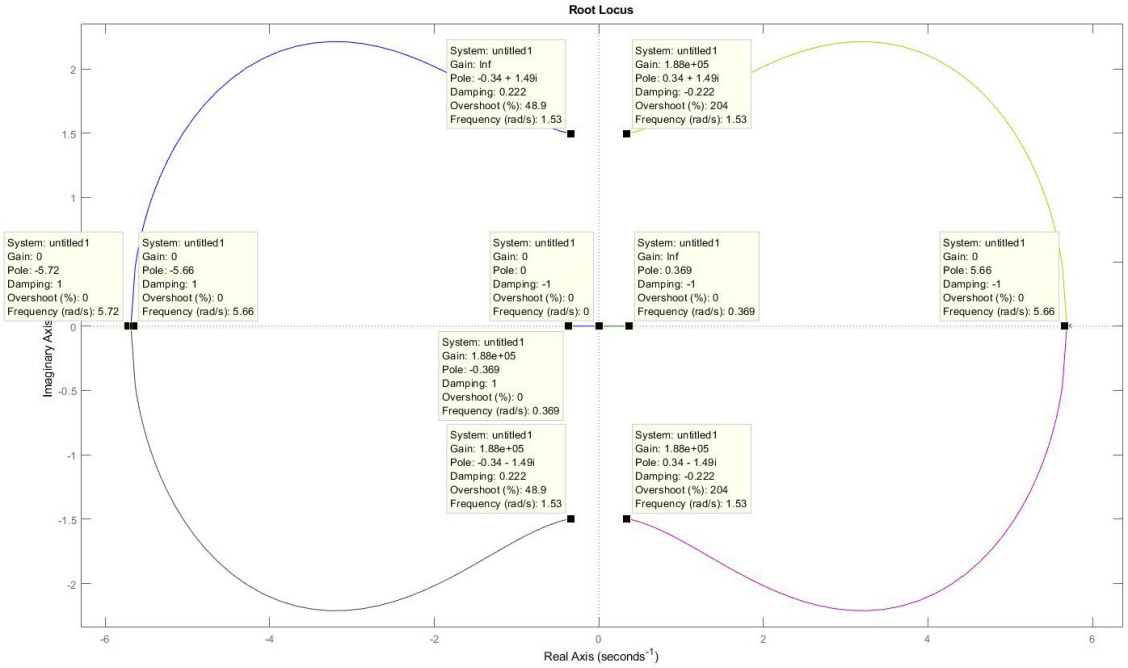

Figure 23: The symmetric root locus of the system controlled by LQR [49, Section 4.7].

In Figure 23, the SRL, with its approximate pole location information presented, is drawn around the origin, as there is nothing happening around negative and positive value of 475 along the real axis where one pole exists in the LHP of the continuous-time domain (-475). The new graphs of motor input voltage and body angle based on the LQR method are seen in Figure 24 [49, Section 4.7].

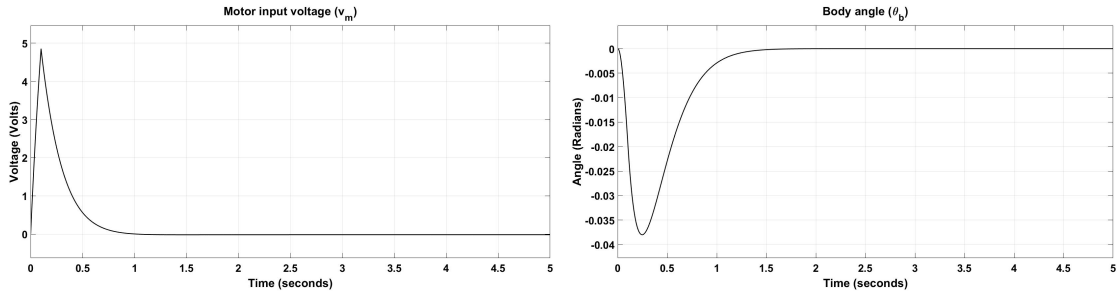

Figure 24: The motor input voltage and robot body angle [49, Section 4.7].

With the LQR method, there is no undershoot in the motor input voltage output and the stabilizing time is three times faster, settling only in one second and there is no overshoot in the body angle output as it settles in 1.5 seconds, which is half the amount of time compared to the second order approximation method seen in Figure 22. Also the peak magnitude of the motor input voltage is slightly lower and the corresponding body angle deviation at its highest smaller than in the second order approximation method. Thus the LQR makes the system more robust and better than by using the second order approximation to design the controller and is the better controller for future use. The most suitable weights were chosen to be one meter for the wheel position, three meters per second for the wheel velocity, ten radians for the body angle and ten radians per second for the wheel angle velocity, as in (78), which are the same as in the reference [49, Section 4.7]. For the ρ, the value 0.1 was selected, as it gave the SRL poles same locations as were in the original TF of (61) as seen in Figure 23. The poles from the transfer function for the LTI that is controlled with LQR are seen in the denominator of

$$\frac{-7.0316e05(s-0.3689)(s+0.3689)(s^2-0.6802s+2.35)(s^2+0.6802s+2.35)}{s^2(s+475.1)(s-475.1)(s+5.72)(s+5.657)(s-5.657)(s-5.72)}.$$

[49, Section 4.7]

4.2.2 Designing a State Observer

The state observer, e.g. for a wheel position $\widehat{\mathbf{x}}$, is an estimator that helps the design process of arbitrary pole positioning by giving the gains matrix K all the state information needed in the input u for this closed-loop system first in continuous-time domain and later in discrete-time domain. The state information vector $\mathbf{x}$ of the wheel position x_w has to be observed and simulated first before trusting the velocity sensor's measured output $\mathbf{y}_{acc}$ in order to obtain all needed states' information. The simulation, which is the observer, receives the same input as the real system thorough u and outputs an explicitly calculated estimation $\widehat{\mathbf{y}}$, which should be close to the real system output of y and that gives an access for all the states of $\mathbf{x}$. If the observer was perfect, the output of the estimator would be exactly the same as the output of the real system. The design process of a state observer is now possible to be established as we have stabilized the balancing robot system thorough feedback and feedforward methods in the matrices A, B, C and D. The $n \times 1$ type feedback gain matrix L (the full-order observer gain) is used in the observer to multiply the 1×1 type error based

terms of the real and estimated system outputs in order to obtain the simulated environment of the closed-loop system. By implementing the closed-loop observer, a new system of $\dot{\hat{\mathbf{x}}}$ is obtained that takes the output of the real system as its input. In the observer, if the matrix pair of A and C is observable, then the gain matrix L can be used to make the estimated system arbitrary pole placement possible. [41] In order to stabilize or observe a system using state feedback, one needs to know the system states. A direct observation is not always possible, as it is also in the MinSeg's case, so its internal states have to be estimated by looking at the system's output. When the system is observable, then its states can be reconstructed via the output measurements by the state observer. In the case of the observer gain L being very high, the desirable quick convergence of the estimator output to the real system output is obtained in a Luenberger observer. The gains of both matrices K and L can be chosen independently without risking the system's stability. [40] The poles of the observer $(A - LC)$ should converge faster than the poles of the system $(A - BK)$ and we are going to have from two up to six times faster observer than the controller. The $\tilde{A}$ denotes the estimate error of the matrix A acquired empirically from the system. The full-order estimator ($\dot{\hat{\mathbf{x}}} = \mathbf{F}\hat{\mathbf{x}} + \mathbf{G}u$) estimates the actual state $\mathbf{x}$, but contains redundancies because it estimates directly measured state-variables. [17, Section 7.7] The reduced-order estimator reconstructs the state variables that cannot be directly recovered from the outputs of the system [1]. The slower the observer is, the less resilient the controller is to disturbances and the noisier the sensors are, the slower the observer should be [47]. Now that the system is ready to be observed, the two Luenberger observers (a full- and a reduced-order one) can be designed. Both of these observers measure the body angle θ_b and the wheel position x_w and change the output matrix to be as in (79). [49, Section 4.8]

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \Longrightarrow \qquad \begin{bmatrix} y_{x_w} \\ y_{\theta_b} \end{bmatrix} = C \begin{bmatrix} x_w \\ \dot{x}_w \\ \theta_b \\ \dot{\theta}_b \end{bmatrix} \tag{79}$$

In (79), the output matrix C now contains the information needed from the two aforementioned measurements. The full-order Luenberger observer as well as the reduced-order Luenberger observer that will be designed here assume the matrix C parameters described in (79), but whereas the full-order observer uses directly the poles allocation method, the reduced-order observer in addition to that assumes the y_{x_w} measurement accurate while the y_{θ_b} is assumed not accurate. The matrix B relates only to the input v_m and is similar to a single column matrix. For the reduced-order observer gains matrix M_x, the system matrices $A, B, C, (D = 0)$ are written as in (80). [49, Section 4.8]

$$\begin{cases} \dot{\mathbf{x}} & = A\mathbf{x} + Bu \\ \mathbf{y}_{acc} & = C_{acc}\mathbf{x} \\ \mathbf{y}_{\overline{acc}} & = C_{\overline{acc}}\mathbf{x} \end{cases} \tag{80}$$

In (80) the $C_{\overline{acc}}\mathbf{x}$ denotes all the states of the acceleration measurement sensor output that are not obtained in $C_{acc}\mathbf{x}$. The acceleration output matrix C_{acc} is in the real-axis

$\mathbb{R}^{p\times n}$ as in a $p\times n$ form where p denotes the amount of accurate measurements. Then a basis completion matrix V is selected so that an inverse matrix T^{-1} is a proper change of basis

$$T^{-1}=\begin{bmatrix}C_{acc}\\ V\end{bmatrix}$$

and defines

$$\mathbf{x}=T\mathbf{z}$$

in such a way that

$$\mathbf{z}=T^{-1}\mathbf{x}=\begin{bmatrix}C_{acc}\mathbf{x}\\ V\mathbf{x}\end{bmatrix}=\begin{bmatrix}\mathbf{y}_{acc}\\ V\mathbf{x}\end{bmatrix}=\begin{bmatrix}\mathbf{y}_{acc}\\ \mathbf{X}\end{bmatrix}$$

can be obtained. Thus the new system of (81) is reached. [49, Section 4.8]

$$\begin{cases}\dot{\mathbf{z}} &= \widetilde{A}\mathbf{z}+\widetilde{B}u\\ \mathbf{y}_{acc} &= \widetilde{C}_{acc}\mathbf{z}\\ \mathbf{y}_{\overline{acc}} &= \widetilde{C}_{\overline{acc}}\mathbf{z}\end{cases}\qquad with\qquad \begin{cases}\widetilde{A} &= T^{-1}AT\\ \widetilde{B} &= T^{-1}B\\ \widetilde{C}_{acc} &= C_{acc}T\\ \widetilde{C}_{\overline{acc}} &= C_{\overline{acc}}T\end{cases} \tag{81}$$

Which incorporates that

$$\widetilde{C}_{acc}=C_{acc}T=C_{acc}\begin{bmatrix}C_{acc}\\ V\end{bmatrix}^{-1}=\begin{bmatrix}I_p & 0_{p\times n-p}\end{bmatrix}$$

and

$$\mathbf{z}=T^{-1}\mathbf{x}=\begin{bmatrix}\mathbf{y}_{acc}\\ V\mathbf{x}\end{bmatrix}=\begin{bmatrix}\mathbf{y}_{acc}\\ \mathbf{X}\end{bmatrix}$$

and

$$\widetilde{C}_{acc}=\begin{bmatrix}I & 0\end{bmatrix}$$

and

$$\widetilde{C}_{\overline{acc}}=\begin{bmatrix}\widetilde{C}_y & \widetilde{C}_x\end{bmatrix}.$$

So now the new system can be rewritten as in (82), where the first and third subsystems are containing only partial information, while the second subsystem is noninformative and can be removed [49, Section 4.8].

$$\begin{cases}\begin{bmatrix}\dot{\mathbf{y}}_{acc}\\ \dot{\mathbf{X}}\end{bmatrix} &= \begin{bmatrix}\widetilde{A}_{yy} & \widetilde{A}_{yx}\\ \widetilde{A}_{xy} & \widetilde{A}_{xx}\end{bmatrix}\begin{bmatrix}\mathbf{y}_{acc}\\ \mathbf{X}\end{bmatrix}+\begin{bmatrix}\widetilde{B}_y\\ \widetilde{B}_x\end{bmatrix}u\\ \mathbf{y}_{acc} &= \begin{bmatrix}I & 0\end{bmatrix}\begin{bmatrix}\mathbf{y}_{acc}\\ \mathbf{X}\end{bmatrix}\\ \mathbf{y}_{\overline{acc}} &= \begin{bmatrix}\widetilde{C}_y & \widetilde{C}_x\end{bmatrix}\begin{bmatrix}\mathbf{y}_{acc}\\ \mathbf{X}\end{bmatrix}\end{cases} \tag{82}$$

The reduced subsystem derived from (82) is presented here in (83) [49, Section 4.8].

$$\begin{cases} \begin{bmatrix} \dot{\mathbf{y}}_{acc} \\ \dot{\mathbf{X}} \end{bmatrix} & = \begin{bmatrix} \widetilde{A}_{yy} & \widetilde{A}_{yx} \\ \widetilde{A}_{xy} & \widetilde{A}_{xx} \end{bmatrix} \begin{bmatrix} \mathbf{y}_{acc} \\ \mathbf{X} \end{bmatrix} + \begin{bmatrix} \widetilde{B}_y \\ \widetilde{B}_x \end{bmatrix} u \\ \mathbf{y}_{\overline{acc}} & = \begin{bmatrix} \widetilde{C}_y & \widetilde{C}_x \end{bmatrix} \begin{bmatrix} \mathbf{y}_{acc} \\ \mathbf{X} \end{bmatrix} \end{cases} \tag{83}$$

Which in turn can be presented as in (84) [49, Section 4.8].

$$\begin{cases} \dot{\mathbf{y}}_{acc} & = \widetilde{A}_{yy}\mathbf{y}_{acc} + \widetilde{A}_{yx}\mathbf{X} + \widetilde{B}_y u \\ \dot{\mathbf{X}} & = \widetilde{A}_{xy}\mathbf{y}_{acc} + \widetilde{A}_{xx}\mathbf{X} + \widetilde{B}_x u \\ \mathbf{y}_{\overline{acc}} & = \widetilde{C}_y\mathbf{y}_{acc} + \widetilde{C}_x\mathbf{X} \end{cases} \tag{84}$$

Or as in (85) [49, Section 4.8].

$$\begin{cases} \dot{\mathbf{X}} & = \widetilde{A}_{xx}\mathbf{X} + \left(\widetilde{A}_{xy}\mathbf{y}_{acc} + \widetilde{B}_x u\right) \\ \begin{bmatrix} \dot{\mathbf{y}}_{acc} - \widetilde{A}_{yy}\mathbf{y}_{acc} - \widetilde{B}_y u \\ \mathbf{y}_{acc} - \widetilde{C}_y\mathbf{y}_{acc} \end{bmatrix} & = \begin{bmatrix} \widetilde{A}_{yx} \\ \widetilde{C}_x \end{bmatrix} \mathbf{X} \end{cases} \tag{85}$$

Here we can see the transformation from the full-order Luenberger observer of

$$\dot{\hat{\mathbf{x}}} = A\hat{\mathbf{x}} + Bu + L(\mathbf{y} - C\hat{\mathbf{x}})$$

to the reduced-order Luenberger observer of

$$\dot{\widehat{\mathbf{X}}} = \widetilde{A}_{xx}\widehat{\mathbf{X}} + (\widetilde{A}_{xy}\mathbf{y}_{acc} + \widetilde{B}_x u) + L\left(\begin{bmatrix} \dot{\mathbf{y}}_{acc} - \widetilde{A}_{yy}\mathbf{y}_{acc} - \widetilde{B}_y u \\ \mathbf{y}_{\overline{acc}} - \widetilde{C}_y\mathbf{y}_{acc} \end{bmatrix} - \begin{bmatrix} \widetilde{A}_{yx} \\ \widetilde{C}_x \end{bmatrix} \widehat{\mathbf{X}} \right),$$

which contains the new system matrices multiplied by L that can be divided into two parts as in

$$L = \begin{bmatrix} L_{acc} & L_{\overline{acc}} \end{bmatrix}.$$

Thus the expressions can be rewritten again to a form of

$$\begin{aligned} \dot{\widehat{\mathbf{X}}} = \left(\widetilde{A}_{xx} - L_{acc}\widetilde{A}_{yx} - L_{\overline{acc}}\widetilde{C}_x\right)\widehat{\mathbf{X}} \\ + \left(\widetilde{B}_x - L_{acc}\widetilde{B}_y\right)u + \left(\widetilde{A}_{xy} - L_{acc}\widetilde{A}_{yy} - L_{\overline{acc}}\widetilde{C}_y\right)\mathbf{y}_{acc} \\ + L_{\overline{acc}}\mathbf{y}_{\overline{acc}} + L_{acc}\dot{\mathbf{y}}_{acc} \end{aligned}$$

Because the term $\dot{\mathbf{y}}_{acc}$ can introduce numerical problems in MATLAB, a new state of

$$\widehat{\mathbf{X}} = \widehat{\mathbf{X}}' + L_{acc}\mathbf{y}_{acc}$$

is introduced and the dynamics of the observer are written again as

$$\begin{bmatrix} L_{acc} & L_{\overline{acc}} \end{bmatrix}.$$

Thus the expressions can be rewritten again to a form of

$$\dot{\widehat{\mathbf{X}}}' = \left(\widetilde{A}_{xx} - L_{acc}\widetilde{A}_{yx} - L_{\overline{acc}}\widetilde{C}_x\right)\widehat{\mathbf{X}} + \left(\widetilde{B}_x - L_{acc}\widetilde{B}_y\right)u + \left(\widetilde{A}_{xy} - L_{acc}\widetilde{A}_{yy} - L_{\overline{acc}}\widetilde{C}_y\right)\mathbf{y}_{acc} + L_{\overline{acc}}\mathbf{y}_{\overline{acc}} \quad ,$$

where

$$\widehat{\mathbf{X}} = \widehat{\mathbf{X}}' + L_{acc}\mathbf{y}_{acc}$$

and to

$$\widehat{\mathbf{x}} = T\begin{bmatrix}\mathbf{y}_{acc}\\ \widehat{\mathbf{X}}\end{bmatrix},$$

where the coordinates are changed in order to go back to the original space of

$$\mathbf{z} = T^{-1}\mathbf{x}.$$

In a compact form, this is presented as

$$\dot{\widehat{\mathbf{X}}}' = M_1\widehat{\mathbf{X}} + M_2u + M_3\mathbf{y}_{acc} + M_4\mathbf{y}_{\overline{acc}}$$

and

$$\widehat{\mathbf{X}} = \widehat{\mathbf{X}}' + M_5\mathbf{y}_{acc}$$

and

$$\widehat{\mathbf{x}} = M_6\mathbf{y}_{acc} + M_7\widehat{\mathbf{X}},$$

where the M_x denotes the reduced-order Luenberger observer gains. The estimated values of the body angle and wheel position are compared with the real output signal both from the full-order and reduced-order Luenberger observers in Figure 25. [49, Section 4.8]

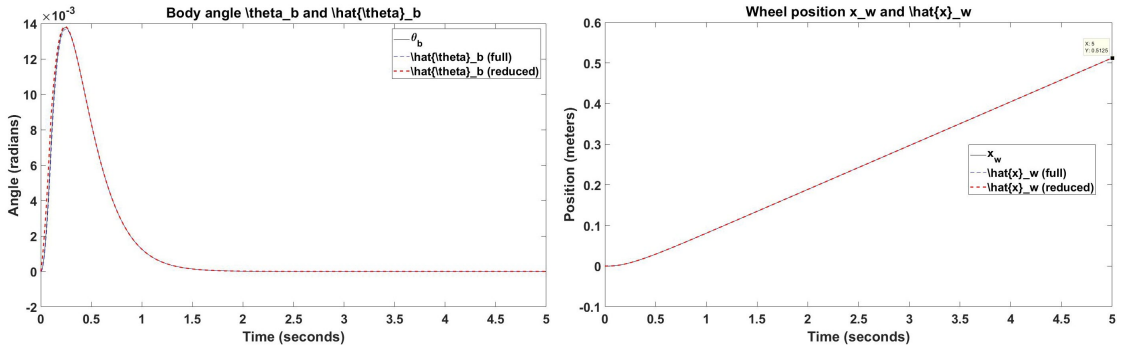

Figure 25: The body angle estimates and wheel position estimates [49, Section 4.8].

By running the file `LabB_ObserverOverSimulator_Continuous_Solution.m`, the values of the continuous-time domain matrix L are given for the 2×4 matrix as

$$\begin{bmatrix} 132.537158335925 & -489.683678722153 \\ -49135.1594113999 & 222167.581809052 \\ -498.447471692429 & 2314.63626755718 \\ 214411.903929445 & -968470.361276375 \end{bmatrix}$$

and the gains of $M_1 ... M_7$ as

$$M_1 = \begin{bmatrix} -2800.63103895759 & -304.50956044142 & 9.13572124000946 \\ -304.519514727064 & -57.456731696725 & 1 \\ 4854.49494611055 & 532.389881514873 & -39.9722293456814 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} 20.57594873876 \\ 0 \\ -90.0275435713545 \end{bmatrix} \quad M_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad M_4 = \begin{bmatrix} 298.410485140821 \\ 57.456731696725 \\ -470.370595644529 \end{bmatrix}$$

$$M_5 = \begin{bmatrix} 2365.59669419524 \\ 304.519514727064 \\ -2951.05545345905 \end{bmatrix} \quad M_6 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad M_7 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

where the maximum error between the real value of the body angle θ_b and the reduced-order estimator is $(0.01382 - 0.01376 =)$ 0.00006 radians for the time signal of 0.238 seconds and $(0.01376 - 0.01367 =)$ 0.00009 radians for the same time instant between the real value of the body angle θ_b and the full-order estimator. For the wheel position the maximum error between the real value of x_w and the reduced-order estimator is zero meters for the time signal of 0.106 seconds and $(0.001142 - 0.0006405 =)$ 0.0005015 meters for the same time instant between the real value of the wheel position x_w and the full-order estimator. [49, Section 4.8] Now we can discretize the controller and observer and add them to the simulator. The discrete analysis is performed with the z-transform [17, Sections 8.2, 8.3]. By running `LabB_ObserverAndControllerOverSimulator_Discrete_Solution.m` that is built from `LabB_ObserverAndControllerOverSimulator_Discrete_Parameters.m`, where all the parameters computed in the two previous steps are put in, and the Simulink file `LabB_ObserverAndControllerOverSimulator_Discrete.slx`, which in turn are constructed of `LabB_ObserverOverSimulator_Discrete_Parameters.m`, where the variables Ad, Bd, Cd and Dd are computed in relation to the original system A, B, C, D, the variable Ld relative to the gains of the full-order discrete Luenberger observer is calculated, the variables $Md1 ... Md7$ relative to the gains of the reduced-order discrete Luenberger observer are computed and these gains are loaded into the PID controller variables kI, kP, kD, and the Simulink file `LabB_ObserverOverSimulator_Discrete.slx`, which are constructed by the MATLAB file `LabB_ControllerOverSimulator_Discrete_Parameters.m`, where the value for K_d is stored in the variable Kd and also the Simulink file `LabB_ControllerOverSimulator_Discrete.slx`, the simulation of the system can be evaluated in order to confirm that the design is suitable for real life application of the balancing robot. The continuous-time system $(A, B, C, (D = 0))$ gives a discrete equivalent of $(A_d, B_d, C_d, 0)$ thorough the MATLAB command $c2d$. The pole mapping to the discrete controller is organized via $z = e^{p\Delta}$, where Δ corresponds to the sampling interval for discrete-time systems. The gains of $K_d, L_d, M_{d1} ... M_{d7}$ are calculated with the MATLAB command *acker* and the simulations are checked for

proper behavior. The Ld gave

$$\begin{bmatrix} 0.214788757084117 & 0.00295444947643281 \\ -0.161493612443987 & 0.649032473569472 \\ 0.0211462353107086 & 0.439250680436783 \\ 1.25095244796608 & 2.60984983511196 \end{bmatrix}$$

as its values, the $M_{d1}...M_{d7}$ gave values as for the reduced-order gains as

$$Md_1 = \begin{bmatrix} 0.0389158565351321 & -0.108182181747258 & 0.0160731852855166 \\ -0.0992409471840242 & 0.771791289927428 & 0.00211481558498821 \\ 1.14215679321497 & 0.165284259340721 & 0.762076740519028 \end{bmatrix}$$

$$Md_2 = \begin{bmatrix} 0.0363630799389995 \\ -0.0178151364972501 \\ -0.541463990215157 \end{bmatrix}$$

$$Md_3 = \begin{bmatrix} -19.2264739040308 \\ -47.5903833125884 \\ -1030.59389999055 \end{bmatrix} \qquad Md_4 = \begin{bmatrix} 0.102333856564175 \\ 0.234081103833142 \\ 0.301960763573873 \end{bmatrix}$$

$$Md_5 = \begin{bmatrix} 19.2264739040308 \\ 47.5903833125884 \\ 1030.59389999055 \end{bmatrix} \qquad Md_6 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad Md_7 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

What these discretized numbers tell as well as in the continuous case, is that the both estimators are successfully discretized for both time domains, even if their results are completely different excluding the matrices $Md6$ and $Md7$. The resulting graphs can be seen in Figure 26. [49, Section 4.9]

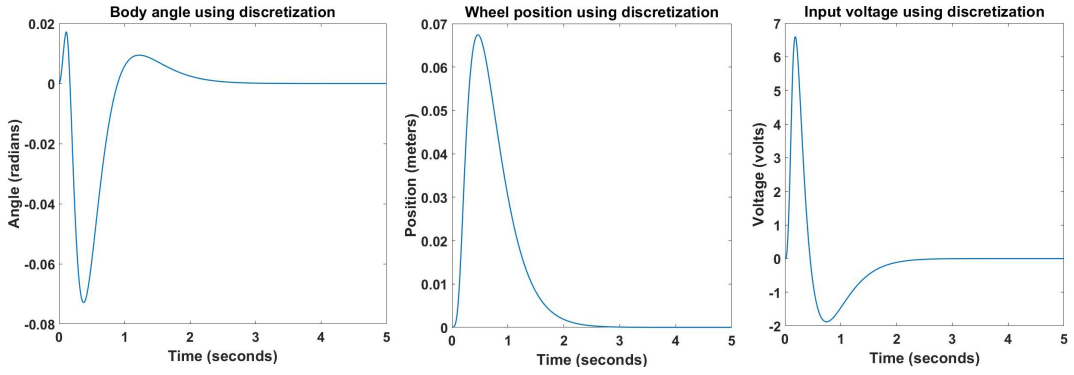

Figure 26: Using discretized controller and observer [49, Section 4.9].

In Figure 26, comparing the results to the undiscretized graphs of Figure 22, now there's slightly more oscillatory behavior, which seems to be a result of the discretization. The wheel position in turn seems to stabilize back to the equilibrium in contrast with Figure 25, where a slight drift from the equilibrium was apparent as in Figure 14, which meant that the system was still unstable, but which is now stabilized with the discrete controller and observer [49, Section 4.9].

4.3 Field-Testing the Improved PID Controller

Now that the simulations are ok, we can put the PID to a field-test. By running `LabB_ObserverAndControllerOverRobot_Solution.m`, which is made of the file `LabB_ObserverAndControllerOverRobot_Parameters.m` that has the parameters from `LabB_ObserverAndControllerOverSimulator_Discrete_Parameters.m`, the needed parameters will be stored into the MATLAB workspace. After the Simulink file `LabB_ObserverAndControllerOverRobot.slx` has been opened, the different observers can be examined when the observer block is clicked. In the observer block, there are the full- and reduced-order Luenberger observers, but besides those, also a numerical observer. The numerical observer works by assuming all of the measurements and measurement derivatives accurate. On top inside the block, there are also two switches that enable a selection of which observer to use. The field-test can now begin. If there was a need to recalibrate the robot, it was done by following the instructions of Chapter 3.4. After recalibration, the robot was switched to battery mode, its USB connected, and the file `LabB_ObserverAndControllerOverRobot_Solution.m` run to begin the field-test and then the file `LabB_ObserverAndControllerOverRobot.slx` was run and after that the robot was held upright and let loose. If the robot started to wander off, the reset button was pressed on the robot and when in battery mode, the USB was disconnected. To troubleshoot, the line (`mean(diff(aafProcessedInformation(MEASURED_THETA_B_INDEX,:))/fSamplingPeriod)` and the text arduino were added to the beginning of the file `LabB_ObserverAndControllerOverRobot_Solution.m`. In MATLAB console, to see the connection details (Arduino MinSeg properties) and to automatically update the server code into the Arduino board, the command arduino gave the properties obtained from the MinSeg and they were as follows: $Port :' COM3'$, $Board :' Mega2560'$, $AvailablePins : 'D2 - D53','A0 - A15'$ and $Libraries : 'I2C','Servo','SPI'$. By running `LabB_TuneTheGyro_SaveParameters.m`, and holding the robot at equilibrium, the GyroBias.mat could be tuned. To tune the fGyroBias variable so that the robot would go forward faster, the robot would have to be held leaning more backward (the battery back side) in the calibration phase when the file `LabB_TuneTheGyro_SaveParameters.m` was ran. To start the saving of the robot parameters into the Arduino board, the unchanged version of the original file `LabB_ObserverAndControllerOverRobot.slx` was copied and the switches turned to the numerical, full-state or reduced-state observer mode. When the file `LabB_ObserverAndControllerOverRobot.slx` was run, the original had to be copied over the ran file in order to change the switches to another setting for another controller version. Then the robot was connected to the computer by its USB and the file `LabB_ObserverAndControllerOverRobot_Solution.m` was ran with the 5V USB mode on the robot and after the (Ctrl+B) was pressed when the Simulink prompted and the compiling finished in MATLAB, the USB could be disconnected from the robot and the battery mode switched on. After this, the MinSeg was put on a clean laminate floor with measuring tape stretched out next to it on the floor. The black Reset button on the robot had to be pressed multiple times before the robot was set loose, in order for the robot start to move only after it was positioned straight

up. If the robot was not straight up before the balancing started, it fell down. When the balancing started successfully, a timer was started on the computer keyboard button that operated under Logitech Gaming software and thus the balancing time was recorded. If (Ctrl+B) was not pressed before the communication started after the set 10 second coundown timer finished, MATLAB would crash, as it didn't change its state from busy to Finished anymore, thus MATLAB had to be restarted. With fGyroBias at -268 by a calibration that was done on top of an uneven table and with the full-state observer, the robot could not balance at all, but the wheels were turning. For the reduced-state observer, the wheels turned once every ten seconds or so, so the robot was almost completely dead, thus there basically was no contest for the balancing field-test. After this observation, only the numerical observer was used in the forthcoming field-test runs. The numerical observer worked the best giving a run time of 3 minutes and 55 seconds, while the robot moved 140 centimeters forward (to its face side) in that time before running into a dumbbell and falling on its back straight on the floor. Hopefully nothing was broken as the event was so sudden and unpredictable that there was no time to prevent the smash. The robot's right wheel corner only slightly touched the neoprene surface of the weight and immediately fell down. With fGyroBias at -148 and the numerical observer, the robot wandered backwards for 1 minute and 55 seconds for the distance of 143 centimeters before falling on its face after a sudden balancing forth-back movement in which the operator again did not see happening and could not prevent the falling. The motor cable took some of the energy out of the falling, bent a little, but was straightened out. When the robot was flat on its face on an uneven table edge, the fGyroBias set to -223. This made the robot wander backwards for 142 centimeters and 3 minutes 30 seconds before falling on its face side, but this time and in all of the following tests, the operator was alert enough to prevent the falling by hands. When the robot was flat on its back near the mat edge, while the wheels didn't touch the floor and the battery pack was not on top of the mat edge, which was estimated by visual inspection to be approximately 0.5 to 1.0 millimeter higher than the mat itself, the fGyroBias was -260. With this setting, the robot wandered backwards for 140 centimeters and 2 minutes 30 seconds before suddenly cutting all power off for an unknown reason and falling on its face side. The floor surface, where the running tests were made, was measured level from horizontal x and y directions with an Ironside 120 centimeter 152229 heavy-duty spirit level. The measurement device used was Ironside 5 meter 150046 High Visibility Self Lock tape measure. The fGyroBias calibration was done on the same surface but on the 4 millimeter thick yoga mat that gave the robot wheels the clearance to turn freely and not wander of while in calibration. With fGyroBias being -260, another test was made after the MinSeg was released closer to a vertical stance than in the previous run and this time the robot wandered off forward to 115 centimeters in 7 minutes and started to balance itself between the 115 centimeter and 120 centimeter mark. After 15 minutes when this field-test started, the Thesis worker took a pillow to the backside and a shirt to the front side of the robot to the 110 centimeter and 125 centimeter marks, so that the test could continue and the Thesis worker could go back to the computer and write down these findings. While the robot was balancing itself still between 115

centimeters and 120 centimeters from the field-test starting point, the reason for the 115 centimeter wandering could be that the robot was not perfectly balanced when it was released at the beginning of the test. At 21 minutes and 35 seconds, the robot started to balance itself more aggressively back an forth and crashing into the impact preventing shirt on the floor giving the total balancing time of 21 minutes 39 seconds, which was so good that no more calibrations were done to the robot in order to keep the most successful parameter of the fGyroBias (-260) unchanged. While the Thesis worker was at the computer near this field-test end, the robot might have bumped into the pillow on the backside that was unnoticed by the operator and then went to the unstable region of the controller and crashed. The pillow and the shirt could not have been more than 15 centimeters apart when the robot was left under no attention, because the robot is only 22 centimeters long and if it had fallen 22 centimeters apart from either pillow or the shirt, the impact could not have been prevented. No camera was set up for catching the error causing event, but the test proved satisfactory and the 4.5 hour long field-testing was decided to be finished for this part before continuing again with the re-designed discrete controller in Chapter 5.4, as there is a saying in automation that if it works, don't touch it! The fGyroBias parameter -260 was kept unchanged from now on. It was noticed that the horizontal calibration of the robot was subject to variations of the hand position, which was tricky to measure exactly, and gave different parameters to the fGyroBias each time the calibration was done for the gyroscope. Now that it was found out that the floor calibration while the robot was lying on its back was the best, the similar result was sure to be repeated from another calibration and would probably give the same result. But, as no manual parameter setting technique was discovered, rather that commencing another calibration session to change the fGyroBias parameter was available, this could not be confirmed. Another thing that was still tested was the robot's response to poking. This test was basically a repetition of the 21 minutes 39 seconds long most successful test, but it was not measured by wandering distance or balancing time, because the MinSeg showed similar behavior of balancing immediately on the spot within 5 sentimeter range as in the most successful test. The distance measurements' accuracy is within +5 and -5 centimeters, because of the difficulty to inspect the starting position of the measurement when the robot started to move after it was released, based solely on a visual observation by the Thesis worker. Also a +/- 5 second accuracy was estimated for the test times. The numerical observer field-test results for five tests can be seen from the table here.

Direction	fGyroBias	Measured distance [cm]	Test time mm.ss
Forward	-268	140	03.55
Backward	-148	143	01.55
Backward	-223	142	03.30
Backward	-260	140	02.30
Forward	-260	125	21.39

Another reason for finishing the field-tests was that no hardware fault or fracture could be done to the robot anymore from an impact to the surface because of any unforeseen unsafe event, in case the robot balanced itself for as long as the

batteries were operating until they ran out of energy. When the robot was released for balancing and poked, it was found out that poking the wheels from either side did not unstabilize the robot so that it would fall down. The higher the poke was made, the less force was needed for the robot to fall. One could poke the wheels quite forcefully without making the robot trip. The MinSeg wobbled to its sides when it was balancing itself and could possibly be prevented by attaching the body to the motor rigidly and possibly by also making the wheels stiffer with stiff foam inserts, but none of these possibilities could be realized, because the Thesis worker did not want to make physical changes to the robot so that the whole Thesis wouldn't need to be written all over again from the beginning. When the physical parameters change on the robot, the calculations have to be done again. But perhaps not a whole lot would need to be changed in the calculations as the mathematical model is already quite simplified. Maybe just some parameter changes for the weight of the robot would do. Anyhow, this alternative of making the balancing more robust was left unexplored because of the Thesis agreed completion target time. But to make the robot more robust, one could use good friction slicks in stead of the current general purpose wheels and a servomotor that is more precisely controlled with the pulse width modulation signal controller. For some reason, the continuous graph could not be drawn from the gyroscope or the velocity sensor anymore in real-time, even though plots of various attributes of the robot could be drawn not in real-time. Even with scopes set on the Simulink file of `LabB_ObserverAndControllerOverRobot.slx`, Simulink gave an error of Both Serial Transmit block and External mode use Pin number 0. This was because the external mode (Ctrl+B) used the same USB port as the serial port for the scope blocks to draw the graphs in real-time and it resulted in a Serial Transmit block and External mode conflict. One could not change the port because the USB port was the one giving the parameters to the robot and the signal information to the computer, thus even if the robot was calibrated from the sequence of Chapter 3.4, the real-time graphs could not be drawn to a plot window. The reason why the real-time graph drawing worked before was and not anymore was left unknown. There might have been a critical line left uncommented in some of the MATLAB scripts, which was left unnoticed and there had been some automatic Windows 10 updates after the previous real-time graph tests, which might have caused this error by altering MATLAB operation functionality in some way and there was no time reserved to fix this issue for this Thesis. [49, Section 4.10]

5 Re-designing the Improved PID on Discrete-Time

5.1 Introduction

In the last part of this Thesis before conclusions, there are five more task to complete. All the tasks here consist of the same mathematical background as in all of the previous chapters, but with the difference that the calculations are made directly in discrete-time domain in order to save time and make the calculations less tedious than within the previous Chapters. By trial and error, not quite the same parameters were found best for the sampling frequency as in the reference. First the discrete (A_d, B_d, C_d, D_d) equivalent is computed for the original (A, B, C, D) model. After that, the controller is re-designed using the LQR technique. Then the observer is re-designed from the controller obtained in the previous step and the final field-tests are made. To conclude the Thesis, an external reference managing module is also designed for the robot. [49, Section 5.1] When the file `LabC_ObserverAndControllerOverSimulator_Discrete_Solution.m` was run, the discrete-time equivalent of the continuous-time (A, B, C, D) linear robot model is presented in the matrices (A_d, B_d, C_d, D_d), where the gain margins for the poles derived at the same 100 Hertz sampling frequency as in the previous Chapter can be seen here in the matrices:

$$Matrix \quad A_d = \begin{bmatrix} 1 & 0.00360626353094881 & -0.000119843523252944 & 0.000342856067241315 \\ 0 & 0.0837572864556059 & -0.000731184382440498 & 0.0191212534611793 \\ 0 & 0.0718692894061524 & 1.00760088290967 & 0.0185440766830242 \\ 0 & 4.0319186878844 & 0.711754185117387 & 0.9229305904641 \end{bmatrix}$$

$$Matrix \quad B_d = \begin{bmatrix} 0.00077537942759026 \\ 0.0433358040189916 \\ -0.00339922314758829 \\ -0.190698856859397 \end{bmatrix}$$

$$Matrix \quad C_d = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$Matrix \quad D_d = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

where the best sampling frequency of 50.00 Hertz was used [49, Section 5.1].

5.2 Re-designing the Controller with LQR Technique

As already done in Chapter 4.2.1, the matrix W of (86) is assumed similar as in (78) to be weighed accordingly towards the undesirable states and K_d is computed for minimizing the discrete-time performance index as seen in (86) [49, Section 5.2].

$$\mathcal{J} = \sum_{k=1}^{+\infty} \left(\rho \begin{bmatrix} x_w(k) & \dot{x}_w(k) & \theta_b(k) & \dot{\theta}_b(k) \end{bmatrix} W \begin{bmatrix} x_w(k) \\ \dot{x}_w(k) \\ \theta_b(k) \\ \dot{\theta}_b(k) \end{bmatrix} + u^2(k) \right) \mathrm{d}t. \tag{86}$$

In (86), also the same parameters of (78) for ρ are used. By running the file `LabB_ControllerOverSimulator_Discrete_Parameters.m` and setting the sampling interval at $\Delta = 0.211$, and watching the robot balancing itself after 30 second time period when the parameters were loaded in the Simulink file `LabB_ControllerOverSimulator_Discrete.slx`, the robot did not fall. When the LQR controller gain K_d minimizing the cost of (78) was set to $\Delta = 0.212$, the robot fell, thus the best sampling interval was determined to be $\Delta = 0.211$ (211 Hertz). The values differ from the reference by $-.009$ for the not falling behavior and $-.013$ for the falling, which were registered into the file `LabB_Solutions.tex`. [49, Section 5.2]

5.3 Re-designing the Observer

After the K_d is now computed, the full-order Luenberger observer L_d and reduced-order observers $M_{d1}...M_{d7}$ can be computed in MATLAB. Now we can choose the pole locations to be such that the poles of the observers can be up to six times faster in discrete-time domain than the poles of the controller. There is a difference for the fastness definition in continuous-time domain and in discrete-time domain. In continuous time, when a pole is e.g. two times faster, it means that the pole is located two times further along the real axis to the left hand side on LHP. This same phenomenon is not possible in discrete-time, as the pole cannot reside beyond the unitary circle, because this would make it unstable. Hence, a pole at location 2 along the real axis in discrete-time domain would make the system unstable and not faster. [49, Section 5.3] In order to make a discrete-time pole faster, then it holds that

$$p_z = e^{p_s \Delta} \quad \Longrightarrow \quad p_s = \frac{l_n(p_z)}{\Delta}. \tag{87}$$

And if $p_z^{'}$ is x times faster than p_z, it holds that

$$p_z^{'} = e^{p_s^{'} \Delta} = e^{x p_s \Delta} \quad \Longrightarrow \quad x p_s = \frac{l_n(p_z^{'})}{\Delta} \quad \Longrightarrow \quad x \frac{l_n(p_z)}{\Delta} = \frac{l_n(p_z^{'})}{\Delta}, \tag{88}$$

from which $p_z^{'}$ can immeadiately be calculated from p_z as its and x's function. The formula for describing how to make a discrete pole x times faster is presented in equation (89) [49, Section 5.3].

$$p_z^{'} = p_z^x \tag{89}$$

By running `LabC_ObserverAndControllerOverSimulator_Discrete_Solution.m` that calls the *.m files `LabC_Solutions_ComputeFullStateObserver_discrete.m`, `LabC_Solutions_ComputeLQRController_discrete.m` and the *.m file `LabC_Solutions_ComputeReferenceGains.m` and needs the file `LabC_Solutions.m` to store values in the file `LabC_Solutions.tex` and where the different frequencies are input. The parameters for Δ that stabilizes the robot in the Simulink file `LabB_ObserverAndControllerOverSimulator_Discrete.slx` are found to be $\Delta = 0.080$ for stabilizing and $\Delta = 0.085$ for not stabilizing, which half as large a sampling frequency as in the reference. The LQR now seems to work worse than in the previous Section without the observer, because of the delay it causes. [49, Section 5.3]

5.4 Experimenting with the Robot

Now that all the poles have been calculated directly on discrete-time, it can be noted that there was no need for first designing a continuous-time controller and its poles and only after that the discrete poles. Now we can make a field-test to the robot to see how different sampling frequencies affect it in real life. By running the file `LabC_ObserverAndControllerOverRobot_Solution.m` and thus storing the corresponding parameters into the MinSeg's memory, the frequency that made the robot not to fall was $\Delta = 0.01$ and the one that made it fall was discovered to be 20 Hertz ($\Delta = 0.02$). Below here is a table of the field-tests that were arranged in such an order from top to down that the highest line on top of the table indicates the first field-test experiment. The fGyroBias was kept at -260 and same margins for error were valid and the same measurement equipment was used as in the previous field-test runs. [49, Section 5.4]

Direction	fSamplingPeriod	Measured distance [cm]	Test time mm.ss
Forward	0.01	152	05.05
Forward	0.02	60	01.35
Forward	0.015	172	04.55
Forward	0.005	172	02.05

Before the first successful run, the robot was sampled with a value of 85 Hertz and couldn't stabilize itself at all, thus confirming the simulation results. The first successful run showed promising aspects, as the robot rolled back and forth to the point of 150 centimeters until climbing onto the 4 mm thich yoga mat, which was new. The robot didn't fall, but backed off from the mat for a few centimeters and the tried again to roll over onto the mat. But, there was again this neoprene dumbbell in the way and the robot bumped into it next. Then the force of the bump was high enough to twist the wheels to the right and thus altering the robot course of direction by approximately ten degrees clockwise. This maneuver made the robot into an approximately 45 degree angle towards the dumbbell and it started to move back and forth between the floor surface and the dumbbell laying on top of the yoga mat. Then the robot bumped on to the dumbbell with ever increasing force for five bumps until running onto the dumbbell too fast and finally falling on its rear side, when it was interfered with hands in order to prevent the impact onto

the hard laminate surface. The reason why the robot fell and couldn't stabilize, must have been because it was not designed to face a disturbance coming from the wheels and not from the body. The area where the experiments were made was only so small that no conclusive observations could be made, because there was not enough test area to perform a full size field-test available. As observed earlier, no serial link related graphs could be drawn longer than for a one second period, so no enough informative graphs could be plotted for an unknown reason. The real life results gave the knowledge that in simulation, there can be about eight times higher a sampling frequency than in real life, for which the robot could stabilize itself. The third field-test run ended for the robot running into a plastic barbell face front, after which its fall backwards was prevented by hands. Some slight jerking motion and longer back and forth motion distance than in the two previous test runs was noticed. The jerking motion might have been a cause of the five Hertz bigger sampling frequency than in the first successful run. The robot was able to balance itself at 15 Hertz sampling frequency, but only barely. Thus the successful run was determined to be with 10 Hertz sampling frequency. Then the last test was with 5 Hertz, when the robot wheel base from the outer edges of the tires was remeasured in case it had changed, but the results showed that it was still firmly at 7.0 centimeters after all these tests made. In this test, the robot moved slower back and forth but faster over a distance and couldn't stabilize itself either after running into the barbell, but fell backwards into the Thesis writer's hands. It looked like it was going to fall asleep, but then suddenly woke up and balanced itself again and didn't look as perky as in the other runs, which was the reason why the 5 Hertz sampling rate was not selected as best performing, because it's not fast enough.

5.5 Managing External Reference Signals

To manage the reference signals, it is advisable to use such methods that do not interfere with the state observer. In Figure 27, such a method is used as a control configuration, where the reference signal enters the system as a feedforward term, which cancels out the poles of the observer, thus avoiding exciting the state observer [49, Section 5.5].

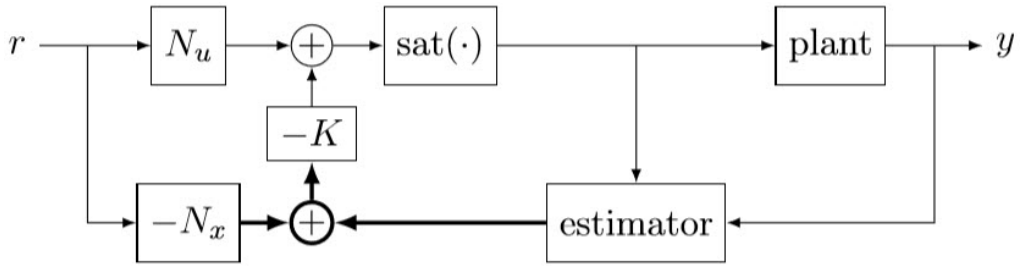

Figure 27: The introduction of the reference input [49, Section 5.5].

Now the values of N_u and N_x are computed in order to make sure that the motor voltage input DC gain from the reference r to output y is kept at 1, as can be seen

in the Figure 27. To ensure this, the equation of (90) is introduced. [49, Section 5.5]

$$\begin{bmatrix} N_x \\ N_u \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \tag{90}$$

In equation (90), the C, which is a row matrix of the type $C = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$, shows that C corresponds to the measurement value of the position of the wheel (x_w), as we are interested of the wheel position reference and not of the angle of the body. But, this equation cannot be used for the discrete-time system, as in discrete-time settings, the DC gain has to be unity and begins from a different equilibrium definition. The equilibrium definition is represented in the equation (91) here. [49, Section 5.5]

$$\begin{aligned} N_{xd} y_{ss} &= A_d N_{xd} y_{ss} + B_d N_{ud} y_{ss} \\ y_{ss} &= C_d N_{xd} y_{ss} + D_d N_{ud} y_{ss} \end{aligned} \tag{91}$$

By running the file `LabC_CompensatorOverRobot_Solution.m`, which is constructed from the file `LabC_CompensatorOverRobot_Parameters.m` and the Simulink diagram of `LabC_CompensatorOverRobot.slx`, the formulas needed for N_u and N_x are derived into the equations of (92) [49, Section 5.5]

$$\begin{bmatrix} A_d & B_d \\ C_d & D_d \end{bmatrix} \begin{bmatrix} N_{xd} \\ N_{ud} \end{bmatrix} = \begin{bmatrix} N_{xd} \\ 1 \end{bmatrix} \implies \begin{bmatrix} N_{xd} \\ N_{ud} \end{bmatrix} = \begin{bmatrix} A_d - I & B_d \\ C_d & D_d \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}, \tag{92}$$

where the numerical values are

$$N_{xd} = \begin{bmatrix} 1, & 0, & 0, & 0 \end{bmatrix}^T \quad and \quad N_{ud} = 0.$$

The conclusion is that the more disturbed the reference signal is, the higher is the probability that the MinSeg cannot stabilize itself and falls [49, Section 5.5].

6 Conclusions

The behavior of an inverted pendulum makes the closed-loop controller design process more involved than designing a controller for e.g. an ordinary pendulum, which can be done in open-loop. The chosen design development assignment was for a MinSeg robot that was supposed to balance on its own with two weels. After the controlling target was selected to be the balancing motion, the work of the controller design could be initiated. By discovering the satisfactory free-body diagram of the robot, the Equations of Motion could be reached and thus the dynamics for the wheel, body and the motor derived by e.g. utilizing the Newton equations for movement. The calculations had to be precise in terms of the EOM in order to obtain a well behaving controller, thus the linearization had to be done around the equilibrium point and the corresponding matrices had to be derived from the linearized equations. Although some reference material was somewhat misleading, a working solution was found for these different parts of the robot design by trial and error. After the linearization, the transfer function was determined and it was transformed into the z domain. Then the disturbance model was added to the robot and the PID was converted into discrete domain for the digital controller to be able to read the input values from a personal computer. After successful simulations, the PID was put thorough a field-test and the correct sequence to establish the communications was obtained. Then the PID was improved with both State-Space and State Observer designs, so that a longer period of balancing could be reached and that the robot would be controlled in a robust stability region. Finally the controller was re-designed directly to discrete domain and to follow an external reference signals. The set goal of making the MinSeg robot balance itself was obtained. The robot has the ability to balance after the parameters are set correctly for the PID, even if poking was involved. The results of the field-tests were expected to be similar as in the reference material, because it was based on an already established and proven concept and this expectation was confirmed correct in the field-tests and with Simulink simulations. The correctness of the laboratory guide was mostly proven by hands-on calculations and the correctness of the field-test results was evaluated by running the solution files from the references and obtaining the correct behavior of the robot and the simulations in return. Some test results could not be obtained from MATLAB because of the inability to plot images in a few solution files. The MATLAB test results were almost identical to the reference material, but because differences in the computer systems hardware and software, all identical results could not be confirmed. Although in general, the robot behaved as expected and the field-tests proved successful. In the end, a more robust controller was designed and thus this Thesis goal was reached, despite different results usually by a margin at most of .1 units of quantity of the reference material. Some Simulink simulations showed similar results for the continuous-time controller as for the discrete-time controller, which were different in the reference material. Also some values simulating the robot disturbance threshold were slightly lower than in the reference, for an unknown reason. Why this happened, was left unclear as the other parameters than the discrete controller gain K_d were input the same in the simulation tests as they were in the reference material, thus one might expect similar

disturbance amount to exist in the simulations in this Thesis. Also the last controller version that was installed to the robot made the accelerometer or the gyroscope or both drift in such a way that the robot started wandering off from the point where it was released. This might have been a result of some line left uncommented in the solution file of lab C, which would have left some computations unmade and thus making the robot not able to locate where its wheels were and thought that the wheels were someplace else than where they were in reality. It was not a cause of uneven surface as the surface was measured even. This hypothesis couldn't be confirmed as the time reserved for more tests was already used at that point. While this Thesis did not explore other areas than what were in the references, in the future applications of MinSeg, one might want to add more features to the robot, such as cell phone controls or autonomous behavior in terms of the robot exploring its surroundings by itself. Also there should be a considerable amount of time reserved for field-testing purposes in order to gain the best results and while running the field-tests, one should make sure that there is enough space for a long-lasting test session and that the robot was handled safely and was kept unharmed.

References

[1] *6.5 State Reconstruction.* Wolfram. [Online]. Available: https://reference.wolfram.com/applications/pcs/FunctionIndex/PoleAssignmentAndStateReconstruction/ReducedOrderEstimator.html [Accessed on 05.08.2018]

[2] *Ackermann's formula.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Ackermann%27s_formula [Accessed on 05.05.2018]

[3] *Angular Velocity Formula.* TutorVista.com. [Online]. Available: https://formulas.tutorvista.com/physics/angular-velocity-formula.html [Accessed on 04.15.2018]

[4] *Bandwidth (signal processing).* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Bandwidth_(signal_processing) [Accessed on 04.29.2018]

[5] *Bode plot.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Bode_plot [Accessed on 04.29.2018]

[6] *Closed-loop transfer function.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Closed-loop_transfer_function [Accessed on 04.30.2018]

[7] *Control Bootcamp: Controllability, Reachability, and Eigenvalue Placement.* Youtube Steve Brunton. [Online]. Available: https://www.youtube.com/watch?v=FLByezr38Ps [Accessed on 05.15.2018]

[8] *Control Systems/Stability.* Wikibooks.org. [Online]. Available: https://en.wikibooks.org/wiki/Control_Systems/Stability [Accessed on 04.25.2018]

[9] *Dean Kamen.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Dean_Kamen [Accessed on 03.30.2018]

[10] *Derivative Rules.* Maths is fun Calculus. [Online]. Available: https://www.mathsisfun.com/calculus/derivatives-rules.html [Accessed on 04.15.2018]

[11] *Digital control.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Digital_control [Accessed on 04.29.2018]

[12] *Dirac delta function.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Dirac_delta_function [Accessed on 04.30.2018]

[13] *Discrete control 2: Discretize! Going from continuous to discrete domain.* Youtube Brian Douglas. [Online]. Available: https://www.youtube.com/watch?v=rL_1oWrOplk [Accessed on 05.17.2018]

[14] *Dividing by Zero.* Maths is fun Numbers. [Online]. Available: http://www.mathsisfun.com/numbers/dividing-by-zero.html [Accessed on 04.25.2018]

[15] *Filter coefficient (N).* MathWorks Documentation: PID Controller, Discrete PID Controller. [Online]. Available: https://se.mathworks.com/help/simulink/slref/pidcontroller.html#br9ejg0-18 [Accessed on 04.30.2018]

[16] *Finding Torque For Angled Forces.* Youtube Khan Academy Physics. [Online]. Available: https://www.youtube.com/watch?v=ZQnGh-t25tI [Accessed on 04.15.2018]

[17] Franklin, G., Powell, J. and Emami-Naeini, A. *Feedback Control of Dynamic Systems.* Seventh Edition. Pearson, 2015.

[18] *Free body diagram.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Free_body_diagram [Accessed on 03.28.2018]

[19] *Free Body Diagram Sine And Cosine Components.* Youtube Professor Matt Anderson. [Online]. Available: https://www.youtube.com/watch?v=KgOqTB1MIKk [Accessed on 04.15.2018]

[20] *How to Multiply Matrices.* Maths is fun Algebra. [Online]. Available: https://www.mathsisfun.com/algebra/matrix-multiplying.html [Accessed on 04.18.2018]

[21] *Intro to Control - 12.3 Root Locus Basics Part 1.* Youtube Katkimshow. [Online]. Available: https://www.youtube.com/watch?v=-eaJDo9Bvy8 [Accessed on 04.27.2018]

[22] *Inverse of a Matrix.* Maths is fun Algebra. [Online]. Available: http://www.mathsisfun.com/algebra/matrix-inverse.html [Accessed on 04.18.2018]

[23] *Kirchoff's voltage law (KVL).* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Kirchhoff's_circuit_laws#Kirchhoff's_voltage_law_(KVL) [Accessed on 04.15.2018]

[24] *Lead-Lag Compensator.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Lead%E2%80%93lag_compensator [Accessed on 04.14.2018]

[25] *LQR Method (Dr. Jake Abbott, University of Utah).* Youtube JJAbbottatUtah. [Online]. Available: https://www.youtube.com/watch?v=St5L-ekOKGA [Accessed on 05.06.2018]

[26] *Mechanical System.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Mechanical_system [Accessed on 03.28.2018]

[27] *Moment of Inertia.* Youtube Khan Academy Physics. [Online]. Available: https://www.youtube.com/watch?v=lwOOV5FitAo [Accessed on 03.30.2018]

[28] *Multiplying by the identity.* Purplemath.com. [Online]. Available: http://www.purplemath.com/modules/mtrxmult3.htm [Accessed on 04.25.2018]

[29] *Newton's laws of motion.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Newton's_laws_of_motion [Accessed on 03.28.2018]

[30] *Nyquist–Shannon sampling theorem.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem [Accessed on 04.29.2018]

[31] *Parametric Equation of a Circle.* Math Open Reference. [Online]. Available: https://www.mathopenref.com/coordparamcircle.html [Accessed on 04.15.2018]

[32] *Polar and Cartesian Coordinates.* Maths is fun. [Online]. Available: http://www.mathsisfun.com/polar-cartesian-coordinates.html [Accessed on 04.15.2018]

[33] *Positive-definite matrix.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Positive-definite_matrix [Accessed on 05.06.2018]

[34] *S-plane.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/S-plane [Accessed on 04.25.2018]

[35] *Sampling (signal processing).* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Sampling_(signal_processing) [Accessed on 04.29.2018]

[36] Shui-Chun, L. and Ching-Chih, T. *Development of a Self-Balancing Human Transportation Vehicle for the Teaching of Feedback Control.* IEEE Transactions On Education, Vol. 52, NO. 1, February 2009.

[37] *Siirtyminen tilaesitystapojen välillä.* Säätötekniikan matematiikan verkkokurssi, AS-74.1102. [Online]. Available: http://autsys.aalto.fi/pub/control.tkk.fi/Kurssit/Verkkokurssit/AS-74.1102/Teoria/dynaaminen/dyn-siirtyminen.html [Accessed on 05.15.2018]

[38] *Singular Value.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Singular_value [Accessed on 04.25.2018]

[39] *Sohcahtoa.* Maths is fun Algebra. [Online]. Available: https://www.mathsisfun.com/algebra/sohcahtoa.html [Accessed on 04.15.2018]

[40] *State Observer.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/State_observer [Accessed on 05.08.2018]

[41] *State Observers (Dr. Jake Abbott, University of Utah).* Youtube. [Online]. Available: https://www.youtube.com/watch?v=Jn9ZTxl1vdY [Accessed on 05.08.2018]

[42] *State-space representation.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/State-space_representation [Accessed on 04.16.2018]

[43] *Taylor series.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Taylor_series [Accessed on 04.16.2018]

[44] *The Lego Mindstorms 2.0 servomotor.* Lego Shop. [Online]. Available: https://shop.lego.com/en-GB/Interactive-Servo-Motor-9842 [Accessed on 04.16.2018]

[45] *Unit Circle.* Maths is fun Geometry. [Online]. Available: https://www.mathsisfun.com/geometry/unit-circle.html [Accessed on 04.15.2018]

[46] Varagnolo, D. *R7003E 2017 LP2 - Automatic Control, Labs: Matlab and Simulink code.* Luleå University of Technology, 2018. [Online]. Available: https://staff.www.ltu.se/~damvar/Classes/R7003E-2017-LP2/Code.zip [Accessed on 04.16.2018]

[47] Varagnolo, D. *R7003E LP2 2016, Lessons: Lesson 15, video 1.* Luleå University of Technology, 2018. [Online]. Available: https://staff.www.ltu.se/~damvar/Classes/R7003E-2016-LP2/Lessons/L15-1.mp4 [Accessed on 05.08.2018]

[48] Varagnolo, D. *R7003E 2017 LP2 - Automatic Control, raw video links.* Luleå University of Technology, 2018. [Online]. Available: https://staff.www.ltu.se/~damvar/R7003E-2017-LP2.html [Accessed on 04.11.2018]

[49] Varagnolo, D. and Lucchese, R. *R7003E labs: state-space control of a balancing robot.* Version 1.2.1. Luleå University of Technology, 2016. [Online]. Available: https://staff.www.ltu.se/~damvar/Classes/R7003E-2016-LP2/LabManual.pdf [Accessed on 03.28.2018]

[50] Varagnolo, D. *R7003E labs: state-space control of a balancing robot, Section 9.11: Datasheet.* Version 1.2.1. Luleå University of Technology, 2016. [Online]. Available: https://staff.www.ltu.se/~damvar/Classes/R7003E-2016-LP2/LabManual.pdf [Accessed on 04.18.2018]

[51] *Z-transform.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Z-transform [Accessed on 04.29.2018]

[52] *Zero-order hold.* Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Zero-order_hold [Accessed on 04.29.2018]