Deep Learning and Regular Control Methods

A Performance Evaluation on a Unicycle

Master's thesis in System Control and Mechatronics

MARÍA HELGA JÓNSDÓTTIR

FILIP PETERSSON

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019:06

Deep Learning and Regular Control Methods

A Performance Evaluation on a Unicycle

MARÍA HELGA JÓNSDÓTTIR
FILIP PETERSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems Control and Mechatronics
Chalmers University of Technology
Gothenburg, Sweden 2019

Deep Learning and Regular Control Methods
A Performance Evaluation on a Unicycle
María Helga Jóndsdóttir
Filip Petersson

Supervisor: Tobias Olsson, Combine Control Systems
Examiner: Torsten Wik, Department of Electrical Engineering

Master's Thesis 2019:06
Department of Electrical Engineering
Division of Systems Control and Mechatronics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A picture of the unicycle used in this thesis constructed by the authors.

Typeset in LaTeX
Gothenburg, Sweden 2019

Deep Learning and Regular Control Methods
A Performance Evaluation on a Unicycle
María Helga Jóndsdóttir
Filip Petersson
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In this thesis the differences between control using deep learning and regular control algorithms are highlighted. Moreover, it is pursued to demonstrate how algorithms developed in a simulated environment manage to stabilize a real physical system. The process of design and construction of a unicycle as a system to compare both methods on is described thoroughly. An infinite horizon Linear Quadratic Regulator (LQR) is selected as a traditional control method and a Proximal Policy Optimization (PPO) algorithm is chosen as the deep learning method. The development of the two methods is described thoroughly and their selection is motivated. Performance evaluation is done on the system using the two methods, both in simulation and practice. The two methods are able to stabilize the system in simulation and also when transferred to the real physical system. The performance in practice is similar to the performance in simulation which indicates that both methods handles the transition to the real physical system fairly well, even though the traditional LQR method outperforms the PPO algorithm in most cases. That being said, the deep learning algorithm shows no sign of uncertain behaviours and leaves a promising room for improvement due to its nonlinear properties. Even though the PPO method showed signs of exploratory behaviour, for systems with a known global optimum the traditional control methods are recommended if applicable, due to its simplicity and robustness.

Keywords: Model based control, reinforced learning, linear-quadratic regulator, unicycle, system control, Proximal policy optimization,

Acknowledgements

First we would like to thank our two supervisors, Torsten Wik at Chalmers University of Technology and Tobias Olsson at Combine Control Systems. Both supervisors have been reachable and helpful at all times, given us advise and showed great faith and interest in our work. We would like to give special thanks to Anders Boström for his guidance in the development of a model by using the wonders of rigid body dynamics. Pär-Love Palm our colleague at Combine also deserves a special thanks for showing great interest in the project, provide help and broaden our vision by providing interesting discussion and sharing his knowledge.

María Helga Jónsdóttir, Filip Petersson, Gothenburg, May 2019

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

Deep learning (DL) is a hot topic in modern society and it is one of the most rapidly growing technical fields today. Its methods can be used in both software and hardware related applications. One of many subjects that could benefit from deep learning is control theory. Therefore, the possibility of improvements in the field using deep learning will likely become important in the control research community. The main advantage over the more traditional control methods is its ability to cope with nonlinearities. It enables implementation of wider range of functions and adaptability to more complex systems [1]. However, there are several difficulties in using DL as a control algorithm, which may induce doubts to the method. Stability as well as performance are essential factors in the industry. One drawback in DL applications is its potential lack of robustness when encountering unexpected conditions. When the testing data differs from the training data, DL algorithms may not only exhibit poor performance, but possibly erroneously assume that the performance is good [2].

The replacement of traditional control with deep learning can therefore be unbeneficial depending on the system and its characteristics. There has been a significant progress in generating control policies in simulated environments [3][4]. Algorithms are capable of solving complex physical control problems in continuous action spaces in a robust way. Even though the tasks are claimed to have real world complexity it is hard to find an example of such high level algorithm in an actual real world application.

For executing a DL algorithm on a system in practice, the success is mostly based on training on the physical system itself or on data generated by it [5][6][7]. Doing so can be time consuming, especially as the system gets more complex and training requires more iterations and data. Furthermore, there exist critical systems that are sensitive to failures and training on the physical system itself may therefore be practically infeasible.

In such cases, a more efficient solution is to train on a simulated system and transfer the policy to the real world. In order to do this, one needs to be able to model the system in a detailed way. There can be found both successful and unsuccessful examples where this is performed [8][9], which may race questions about the actual complexity of the task of transferring a simulated policy to a physical system.

Furthermore, one might wonder if a traditional control method would perform better or worse on the same system. In order to recognize how good the deep learning algorithm actually is performing, One would like to benchmark another method on a similar control level. That kind of comparison is hard to find.

Due to its unstable equilibrium point, the inverted pendulum setup is a commonly used benchmark in control theory [10]. There are many variations of this system all based on the same principal dynamics. One example is a unicycle, which principal dynamics can be viewed as an inverted pendulum in two dimensions [11].

1.2 Purpose

In this thesis project, a unicycle was to be designed and stabilized around its unstable equilibrium point using both a traditional control method and a DL algorithm trained in simulation and transferred to the hardware. The main purpose is to provide an example of a fair comparison between the two methods on a benchmark control problem. The level of performance and robustness of both the traditional control method and a state of the art DL algorithm was to be compared in simulation and in practice. Finally, the thesis should also give a comparison of how the performance of the designed control algorithms differs in simulation, to that of a real physical system.

1.3 Ethical Aspects

With the immense amount of information available online, one needs to ensure that the things contributed follow basic ethical principles. This implies not altering any results and not hide a possible flaw (that could later be revealed at an unfortunate time). Even though ethical aspects can be hard do find in some work it is important to pay attention to them and be aware of that future work based on ones work can be used in unethical situations.

This thesis will provide an example of how modern DL algorithms an act as an alternative to traditional control algorithms. It is important that results are true and clear in the case of this work being used as a foundation for further development of a control for a system that could serve a different purpose and ensure that no harm could be done that can be traced to the control algorithm.

1.4 Outline

In Chapter 2, a brief overview of the theory used throughout the thesis is presented. The aim of this chapter is to summarize definitions of necessary concepts in one place. It is left to the reader to further explore more detailed history and proofs of the concepts through the cited references.

In Chapter 3, the design of the unicycle is presented. It starts with the hardware design of the mechanical structure and the electrical system. Once these have been described, a mathematical model is derived and the embedded software system is described. Finally, a design validation is done in order to verify that the mathematical model yields a satisfactory simulation of the constructed unicycle.

Based on the design, the traditional control method and the DL algorithm are chosen and justified in Chapter 4. It gives a detailed description of the development in simulation of both algorithms, which are then transferred to the real physical system.

The results are presented in Chapter 5. This chapter starts off with a detailed test procedure to make the results replicable. Using this test procedure, the results of the tests are presented. These results are then discussed and analyzed in Chapter 6. First, a general comparison between the traditional control method and the DL algorithm is made. Then, a comparison of how the performance differs from simulation to the real physical system is made for each control algorithm. This is followed by a generalization of the achieved results, as well as suggestions for future work.

Finally, in Chapter 7 the main conclusion of the thesis is presented.

1.5 Open Source Repository

For the interested reader, all the developed software such as the embedded C++ code and the mathematical model can be found at the open source GitHub repository (`https://github.com/filpet95/Unicycle_PPO_vs_LQR`). It is released under a GNU licence, which in short implies that the software is available and free of use but any changes must be available to the public.

2
Theory

In this chapter, a brief overview of the concepts used in the thesis is presented.

2.1 Rotating Coordinates

Consider two arbitrary coordinate frames F_0 and F_1 with identical origin. Any arbitrary vector $\boldsymbol{p^0}$ in the frame F_0 given by

$$\boldsymbol{p^0} = \begin{bmatrix} p_x^0 \\ p_y^0 \\ p_z^0 \end{bmatrix} \tag{2.1}$$

is, in the frame F_1 given by

$$\boldsymbol{p^1} = R_0^1 \begin{bmatrix} p_x^0 \\ p_y^0 \\ p_z^0 \end{bmatrix}, \tag{2.2}$$

where R_0^1 is the rotation matrix from frame F_0 to F_1 [12]. The rotation matrix is orthogonal, which implies that the inverse transformation can be given by

$$R_1^0 = (R_0^1)^{-1} = (R_0^1)^T. \tag{2.3}$$

Three rotation matrices of particular interest are the rotations about each fixed axis, also called the elementary rotations. The rotation about each axis in frame F_0 are then given by

$$R_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & \sin(\varphi) \\ 0 & -\sin(\varphi) & \cos(\varphi) \end{bmatrix}, \quad R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{2.4}$$

$$\text{and} \quad R_z(\psi) = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.5}$$

Combining these rotations in a fixed body system, one can construct more complex rotations. In fact, by using the simple rotations about three consecutive coordinate axes, any rotation can be described given that two consecutive rotations are not done about the same axis. This way to parametrize a rotation is called Euler angles. As matrix multiplication in the general case does not commute, the order in which the elemental rotations are done matters and depends on the system. Hence, using Euler angles, there are 12 different possible sequences that can describe a rotation in 3 dimensions [12].

2.2 Lagrange's Dynamics

Lagrange's Dynamics are a tool used to model mechanical systems. It utilizes a generalized configuration space to find the equations of motion of the system. In general, equations of motion include reaction forces that may enlarge the complexity of the modelling significantly. As the reaction forces are often absent in Lagrange's equations, the number of equations describing the dynamics are often fewer when using Lagrange's dynamics with a good set of generalized coordinates [12][13].

2.2.1 Generalized Coordinates

Generalized coordinates are a set of coordinates that can uniquely describe the position of a system. If the number of generalized coordinates is equal to the degrees of freedom (DOF) of the system, they are called free, which one often strives for when selecting generalized coordinates. If the number of generalized coordinates is larger than the DOF of the system, one can often eliminate coordinates using holonomic constraints in order to achieve free generalized coordinates [12].

2.2.2 Generalized Forces

Consider an arbitrary point p in a system with a position described as a function of the generalized coordinates $\boldsymbol{q}$ and time t according to

$$p(\boldsymbol{q}, t) = \boldsymbol{r}(q_1, \ldots, q_M, t), \tag{2.6}$$

where $\boldsymbol{r}$ is an arbitrary function. The virtual work is defined as the work from an infinitesimal displacement $\delta\boldsymbol{r}$ given by

$$\delta\boldsymbol{r} = \sum_{i=1}^{M} \frac{\partial p}{\partial q_i} \delta q_i \tag{2.7}$$

from a force F under the condition that the time is kept fixed. Given the sum of all external forces $\boldsymbol{F_e}$ acting on p, the virtual work is given by

$$\delta W = \sum_{i=1}^{M} \boldsymbol{F_e} \cdot \frac{\partial p}{\partial q_i} \delta q_i. \tag{2.8}$$

The generalized force acting on p for each generalized coordinate q_i is defined as the coefficient in front of the virtual displacement δq_i in Equation 2.8, i.e.

$$Q_i = \boldsymbol{F_e} \cdot \frac{\partial p}{\partial q_i}, \tag{2.9}$$

where $i = 1, \ldots M$ [12].

2.2.3 Lagrange's Equations

Consider a system with M DOF described by free generalized coordinates $\boldsymbol{q} \in \boldsymbol{R}^M$, i.e.

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_M \end{bmatrix}. \tag{2.10}$$

The equations of motion of this system are given by

$$\frac{d}{dt}\frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial q_i} = Q_i(\boldsymbol{q}), \quad i = 1 \dots M, \tag{2.11}$$

where $Q(\boldsymbol{q})$ are the external forces expressed in generalized coordinates and $\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the Lagrangian defined according to

$$\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = T(\boldsymbol{q}, \dot{\boldsymbol{q}}) - U(\boldsymbol{q}), \tag{2.12}$$

where $T(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the kinetic energy of the system and $U(\boldsymbol{q})$ is the potential energy of the system [12][13].

2.3 Linearization

The dynamics of a nonlinear system described by

$$\frac{d}{dt}\boldsymbol{x}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \qquad \boldsymbol{x} \in \boldsymbol{R}^n, \boldsymbol{u} \in \boldsymbol{R}^m \tag{2.13}$$

$$\boldsymbol{y}(t) = \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \qquad \boldsymbol{y} \in \boldsymbol{R}^p \tag{2.14}$$

can be approximated around an equilibrium point $\boldsymbol{x}_s$,$\boldsymbol{u}_s$ in which

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}_s(t), \boldsymbol{u}_s(t)) = 0 \tag{2.15}$$

as linear dynamics [14]. Provided that $\boldsymbol{f}$ and $\boldsymbol{h}$ are differentiable, the linear dynamics are a valid approximation of the nonlinear system in a local neighborhood around the equilibrium point and are called the linearized system. The dynamics of the linearized system are achieved by expanding $\boldsymbol{f}$ and $\boldsymbol{h}$ around the equilibrium point using the first order Taylor expansion. This will result in the linearized system

$$\begin{aligned} \frac{d}{dt}\Delta\boldsymbol{x}(t) &= A\Delta\boldsymbol{x}(t) + B\Delta\boldsymbol{u}(t) \\ \Delta\boldsymbol{y}(t) &= C\Delta\boldsymbol{x}(t) + D\Delta\boldsymbol{u}(t), \end{aligned} \tag{2.16}$$

where $\Delta\boldsymbol{x}(t) = \boldsymbol{x}(t) - \boldsymbol{x}_s$, $\Delta\boldsymbol{u}(t) = \boldsymbol{u}(t) - \boldsymbol{u}_s$ and A, B, C and D are system matrices given by

$$A = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\substack{\boldsymbol{x}=\boldsymbol{x}_s \\ \boldsymbol{u}=\boldsymbol{u}_s}}, \qquad B = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{\substack{\boldsymbol{x}=\boldsymbol{x}_s \\ \boldsymbol{u}=\boldsymbol{u}_s}}, \qquad C = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right|_{\substack{\boldsymbol{x}=\boldsymbol{x}_s \\ \boldsymbol{u}=\boldsymbol{u}_s}}, \qquad D = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{u}}\right|_{\substack{\boldsymbol{x}=\boldsymbol{x}_s \\ \boldsymbol{u}=\boldsymbol{u}_s}}. \tag{2.17}$$

Commonly, for notation purposes the Δ in Equation 2.16 is left out [14]. Doing so, the linearized system is given by

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= A\boldsymbol{x}(t) + B\boldsymbol{u}(t) \\ \boldsymbol{y}(t) &= C\boldsymbol{x}(t) + D\boldsymbol{u}(t). \end{aligned} \tag{2.18}$$

2.4 Discretization

A continuous time LTI system given by Equation 2.18 can, in discrete time with sample time T_s be approximated according to

$$\begin{aligned} \boldsymbol{x}_{k+1} &= A_d\boldsymbol{x}_k + B_d\boldsymbol{u}_k \\ \boldsymbol{y}_k &= C_d\boldsymbol{x}_k + D_d\boldsymbol{u}_k, \end{aligned} \tag{2.19}$$

where k is the time index representing time instance $t = kT_s$ and A_d, B_d, C_d and D_d are discrete time system matrices determined from the continuous time system. If the input is constant on each time interval $t \in \begin{bmatrix} kT_s & (k+1)T_s \end{bmatrix}$ (so called zero orer hold input), the approximation is exact and the discrete time system matrices are given by

$$A_d = \boldsymbol{e}^{AT_s}, \qquad B_d = \int_0^T T_s\boldsymbol{e}^{A\sigma}d\sigma B, \qquad C_d = C, \qquad D_d = D, \tag{2.20}$$

where $\boldsymbol{e}$ is the matrix exponential [14].

2.5 Infinite Horizon Linear Quadratic Regulator

The infinite horizon linear quadratic regulator (LQR) is a model based state feedback controller. The feedback gain is determined offline from an arbitrary initial state minimizing a weighted sequence of states and inputs over a future time horizon that tends towards infinity. Given two positive, symmetric and semi definite weight matrices $Q_x \geq 0$, $Q_u > 0$ and a linear model, the LQR feedback law is optimal [15].

In the discrete case, the LQR controller is derived as follows. Assume a linear, time invariant (LTI) and controllable discrete time state space model given by

$$\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k, \tag{2.21}$$

where $\boldsymbol{x}$ is the state vector, A and B are constant system matrices with appropriate dimensions and k is the time index. The optimal feedback $\boldsymbol{u}_{opt}(k)$ at time $t = kT_s$, which minimizes the cost function

$$J = \sum_0^\infty \boldsymbol{x}(k)^T Q_x \boldsymbol{x}(k) + \boldsymbol{u}(k)^T Q_u \boldsymbol{u}(k) \tag{2.22}$$

is given by

$$\boldsymbol{u}_{opt}(k) = -K\boldsymbol{x}(k), \tag{2.23}$$

where K is the optimal steady state feedback gain matrix. Given that the system matrices (A, B) are stabilizable, K is determined according to

$$K = (Q_u + B^T\bar{P}B)^{-1}B^T\bar{P}A, \tag{2.24}$$

where $\bar{P}$ is a positive semidefinite matrix, which is the solution of the discrete time algebraic Riccatti equation given by

$$A^T\bar{P}A - \bar{P} + Q_x - A^T\bar{P}B(R + B^T\bar{P}B)^{-1}B^T\bar{P}A = 0. \tag{2.25}$$

By combining Equation 2.21 and 2.23, the closed loop system, i.e.

$$\boldsymbol{x}_{k+1} = (A - BK)\boldsymbol{x}_k, \tag{2.26}$$

has guaranteed phase and gain margins and is asymptotically stable [14].

2.6 Infinite Impulse Response Filters

A continuous time first order standard infinite impulse response (IIR) low pass filter can be described by

$$H(s) = \frac{1}{1 + s\tau}, \tag{2.27}$$

where τ is the time constant. In time domain, it is equivalent to

$$\dot{\hat{y}}(t) = -\frac{1}{\tau}\hat{y}(t) + \frac{1}{\tau}y_m(t), \tag{2.28}$$

where $\hat{y}$ is the estimated state and y_m is the measurement. Discretizing this using forward Euler, one will get

$$\hat{y}_k = (1 - \frac{T_s}{\tau})\hat{y}_{k-1} + \frac{T_s}{\tau}y_{m,k}, \tag{2.29}$$

where k is the time index representing $t = kT_s$ and T_s is the sampling time.

Note that Equation 2.29 will give relatively bad estimations in the first time steps, as the old output will start at 0 while the initial state may be something entirely different. This problem can be solved by adding an adaptive time constant that is low at the beginning and increases with time until it converges to a steady state value. This can be achieved using

$$\begin{aligned} w_k &= \lambda w_{k-1} + 1 \\ \hat{y}_k &= (1 - \frac{1}{w_k})\hat{y}_{k-1} + \frac{1}{w_k}y_{m,k}, \end{aligned} \tag{2.30}$$

where w is initialized to 0. Depending on the tuning constant λ, w will eventually converge to a steady state value w_∞ and then be equivalent to Equation 2.29. By comparing Equation 2.29 and 2.30, one can identify the worst (steady state) time constant by

$$\tau = w_\infty T_s. \tag{2.31}$$

The choice of tuning constant λ is a trade off between slow dynamics and noise on the signal due to the resolution errors.

2.7 Complementary Filter

A complementary filter is a fairly simple alternative to nonlinear filters such as the extended or unscented Kalman filter. The complementary filter fuses two estimations that have complementary properties in the frequency domain [16]. Assume two scalar estimations $\hat{y}_1$ and $\hat{y}_2$, where $\hat{y}_1$ have desirable properties at high frequencies and $\hat{y}_2$ has desirable properties at low frequencies. Furthermore, assume that the estimation $\hat{y}_1$ is obtained by integrating the measurement y_m. The idea of the complementary filter is to apply a high pass filter on the estimation $\hat{y}_1$ and a low pass filter on the estimation $\hat{y}_2$ and fuse the two estimations to one. Doing this, the final estimation $\hat{y}_k$ at the discrete time step $t = kT_s$ is given by

$$\hat{y}_k = (1-\gamma)\hat{y}_{2,k} + \gamma(\hat{y}_{k-1} + T_s y_{m,k}), \tag{2.32}$$

where T_s is the sample time and γ is a tuning parameter. Choosing a γ close to one corresponds to a low cut off frequency and a significant contribution from the estimation with desirable properties at high frequencies [16].

2.8 Kalman Filter

The Kalman filter is widely used in stochastic optimal control and is the closed form solution to the Bayesian filtering equations [17]. Assume a motion and measurement model given by

$$\boldsymbol{x}_k = A_{k-1}\boldsymbol{x}_{k-1} + \boldsymbol{q}_{k-1}, \tag{2.33}$$

$$\boldsymbol{y}_{m,k} = H_k\boldsymbol{x}_k + \boldsymbol{r}_k, \tag{2.34}$$

where $\boldsymbol{x}_k$ is the state, $\boldsymbol{y}_{m,k}$ is the measurement, $\boldsymbol{q}_{k-1} \sim \mathcal{N}(\boldsymbol{0}, Q_{k-1})$ is the process noise and $\boldsymbol{r}_k \sim \mathcal{N}(\boldsymbol{0}, R_k)$ is the measurement noise. Furthermore, assume the prior mean $\boldsymbol{m}_0$ and covariance P_0. Given the measurements $\boldsymbol{y}_{1:k}$, then the optimal linear and Gaussian estimation at time step $t = kT_s$ is given by

$$p(\boldsymbol{x}_k|\boldsymbol{y}_{1:k}) = \mathcal{N}(\boldsymbol{x}_k|\boldsymbol{m}_k, P_k), \tag{2.35}$$

where $\boldsymbol{m}_k$ is the mean and P_k is the covariance of the update step

$$\begin{aligned}
\boldsymbol{m}_k &= \boldsymbol{m}_k^- + K_k\boldsymbol{v}_k, \\
P_k &= P_k^- - K_k S_k K_k^T, \\
\boldsymbol{v}_k &= \boldsymbol{y}_{m,k} - H_k\boldsymbol{m}_k^-, \\
S_k &= H_k P_k^- H_k^T + R_k, \\
K_k &= P_k^- H_k^T S_k^{-1}
\end{aligned} \tag{2.36}$$

and $\boldsymbol{m}_k^-$ is the mean and P_k^- is the covariance of the prediction step

$$\begin{aligned}
\boldsymbol{m}_k^- &= A_{k-1}\boldsymbol{m}_{k-1}, \\
P_k^- &= A_{k-1}P_{k-1}A_{k-1}^T + Q_{k-1}.
\end{aligned} \tag{2.37}$$

Hence, the minimum mean square error estimation at time $t = kT_s$ is $\hat{\boldsymbol{y}}_k = \boldsymbol{m}_k$ [17].

2.9 Reinforcement Learning

The goal of reinforced learning is for an agent to learn the best strategy to react in a certain environment. The reactions from the environment are decided by a model. The model itself is generally unknown by the agent. The agent can stay in one or many states ($x \in S$) of the environment. To move from one state to another the agent takes an action ($a \in A$). Once an action is taken the agent receives a reward from the model as a feedback. The goal for the agent is to maximize the return, which is the expected total cumulative reward. To do so the agent follows a policy ($\pi(x)$) which is a function of states that tells the agent which actions to take in a particular state. Each state has a value function, $V_\pi(x)$ giving the expected future reward for that particular state if the agent follows the policy.

Most reinforced learning algorithms are built on a so called Action-value method. The policy is trained by learning values of actions and selecting actions by their estimated action values [18].

2.9.1 Gradient Decent

In order to get a solution to

$$\frac{\partial L(\theta)}{\partial \theta} = 0, \tag{2.38}$$

one need to find the θ that locally minimizes the function L. If L is a complex function, numerical methods can be used to iteratively update θ in small steps until $\frac{\partial L(\theta)}{\partial \theta}$ is close enough to 0. This can be generally expressed by

$$\theta_{k+1} = \theta_k - \alpha_k \frac{\partial L(\theta_k)}{\partial \theta_k}, \tag{2.39}$$

where θ_k represents the parameter at iteration k, and α_k is a small number that determines the step size at each iteration towards a local minima. In the case of a local maximum each update in Equation 2.39 is instead added to θ_{k+1}. The method is then referred to a gradient ascent. The α_k parameter is often referred to as learning rate in machine learning and can be fixed, adaptive or updated according to a schedule [19].

2.9.2 Multilayer Perceptron

The multilayer preceptron is a linear classifier that classifies outputs to inputs between layers in a neural network. The input is a vector, $\boldsymbol{x}$, what will be multiplied with a vector of weights $\boldsymbol{w}$, and then a bias, b, is added to it as it is passed from one layer to the next. Hence, the output z is given by

$$z = \sum_{i=1}^{n} w_i x_i + b = \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x} + b. \tag{2.40}$$

z is then passed through an activation function yielding the output from the first hidden layers first neuron, $h_1^1 = f(z)$.

Each neuron will have their own set of weights ($\boldsymbol{w}^{\boldsymbol{h_1^1}} : (w_1, w_2, ..., w_m)$). The outputs from all neurons in the first hidden layer ($\boldsymbol{h^1} : (h_1^1, h_2^1, ..., h_n^1)$) will be the input to the next layer,

$$h_1^2 = f(\sum_{i=1}^{n} w_i h_i^1 + b) = f(\boldsymbol{w}^{\boldsymbol{h_1^1 T}} \boldsymbol{h^1} + b), \tag{2.41}$$

where f is an activation function. The same thing is done for all layers until $\hat{y}$ is reached as the output from the final layer (Figure 2.1 [20][21]).

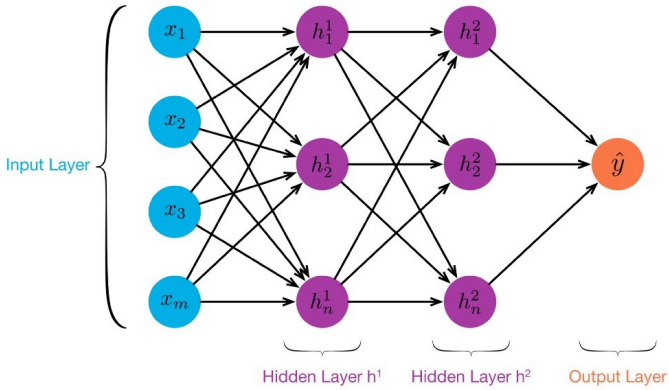


Figure 2.1: A mulitilayer peceptron with an input layer $\boldsymbol{x} : (x_1, x_2, ..., x_n)$, two hidden layers, h^1, h^2, and an output layer $\hat{y}$.

2.9.3 Activation Function

Output from each node in a neural network is usually determined by an activation function. These functions are applied to the outputs from the layers in order to map them to a certain range. In multilayer peceptron a common activation function is the hyperbolic tangent function, defined by

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.42}$$

The function is smooth, bounded and maps the inputs to the range between -1 and 1 [22].

2.9.4 Policy Gradient Methods

The method differs from a more traditional Action-value method mentioned earlier in this section in the way that it does not require the value function to select action, only to learn the policy. It learns the policy directly with a parameterized function with respect to θ,

$$\pi(a|x;\theta) = Pr\{A(t_i) = a|S(t_i) = x, \theta(t_i) = \theta\} \tag{2.43}$$

where $\pi(a|x;\theta)$ is the probability that action a is taken at time t_i given that the environment is in state x at time t_i with parameter θ. Policy gradient methods learn the policy parameter based on the gradient of a scalar performance measure, $L(\theta)$ with respect to the policy parameter. These methods use gradient decent, in order to move θ towards the direction suggested by the gradient $\nabla L(\theta)$ to find the best θ for the policy that produces the highest return [18].

Policy-based methods offer practical ways of dealing with large action spaces, even continuous spaces with an infinite number of actions. Instead of computing learned probabilities for each of the many actions, it can learn statistics of the probability distribution [18].

In general the method starts out with an arbitrary random policy. Then actions are sampled in the environment. If the reward is better then expected the probabilities of taking those actions are increased, if not they are decreased.

2.9.5 Proximal Policy Optimization

This method is built on the policy gradient method, introduced preveously in this section. It uses trust regions to avoid large gradient update. The authors of the method propose an objective function, $L_t(\theta)$, which is approximately maximized in every iteration between an old policy parameter (θ) to a new updated one [23]. This is represented as

$$L_t(\theta) = \hat{E}[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](x_t)], \tag{2.44}$$

where $\hat{E}$ denotes the estimation of emperical average of finite numbers of samples from a batch. c_1 and c_2 are coefficients, S is called entrophy bonus and L_t^{VF} is a squared error loss calculated from the value function,

$$L_t^{VF} = (V_\theta(x) - V_{target})^2. \tag{2.45}$$

The term $L_t^{CLIP}(\theta)$ is a clipped objective,

$$L_t^{CLIP}(\theta) = \hat{E}[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.46}$$

Here the first term is a probability ratio,

$$r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \tag{2.47}$$

between the old policy and the new one. Here $\pi_\theta(a|s)$ is the probability of an action in a given state in a policy. The probability ratio is then multiplied with an estimator of the advantage of this update gives, $\hat{A}_t$. Next term is a clipping operator, which clips the ratio to be within $[1-\epsilon, 1+\epsilon]$. Finally the objective takes the minimal value of the clipped and unclipped ratio creating a bound that prevents the objective function from increasing the policy update to extremes for better rewards [23].

3
Design

In this chapter, the design of the hardware, mathematical model and the software of the system will be described.

The design of the hardware is split up into a mechanical and an electrical part. To begin with, the mechanical structure of the unicycle will be constructed. Then, an electrical construction will be implemented to make the unicycle independent of any external resources.

Once the hardware is in place, a mathematical model will be designed based on the constructed hardware. The purpose of the mathematical model is both to aid the design of the controllers that will be implemented and to simulate the final system for results.

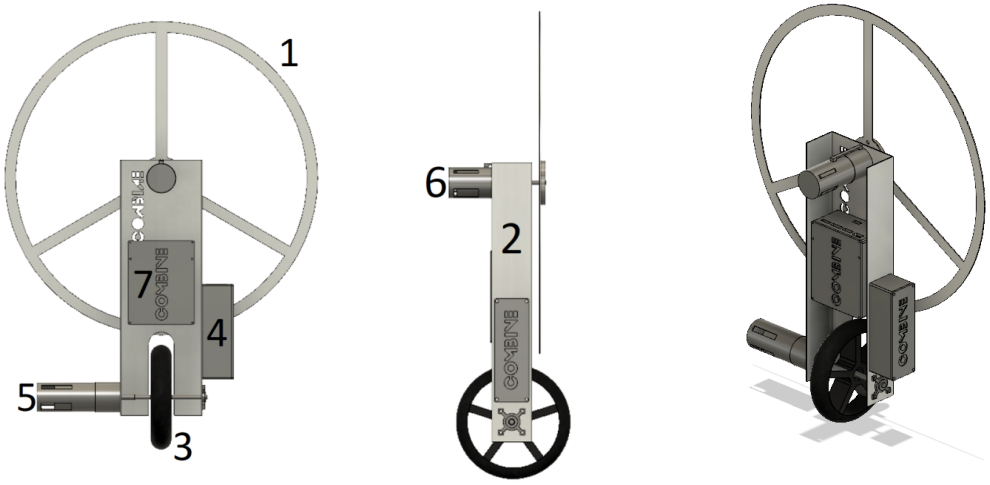
Furthermore, an embedded software algorithm will be implemented. The result of that part will be a complete system to which one can upload a control algorithm to. Hence, all parts from applying input references to sensor readings will be implemented. This includes sensor fusion in order to get accurate readings from the sensor data as well as a working software for each part of the unicycle, which is combined to a complete system.

Finally, in order to validate the designed unicycle with the derived mathematical model, a design validation will be conducted.

3.1 Mechanical Construction

Since the main focus is not on the mechanical aspects in this thesis, the construction is based on a previous design made by Gabriel Pereira Das Nevas [24]. It is a robot that travels on one wheel on the ground. The ground wheel is driven by a motor which task is to prevent the system from falling in the longitudinal direction. Attached to the wheel is a U-shaped aluminum profile which is described as the body. Its task is to combine all components into one rigid body. Finally, on top of the body, a reaction disk driven by another motor is placed. Its task is to prevent the system from falling in the lateral direction.

The reaction disk (1 in Figure 3.1) is made from 1.5 $[mm]$ thick aluminum. It has the outer diameter of 430 $[mm]$ and the inner diameter of 400 $[mm]$ in order to keep



(a) Back view **(b)** Side view **(c)** Viewed from an angle

Figure 3.1: A 3D model of the system created in Fusion360. The main components are the Reaction disk (1), Body (2), Drive wheel (3), Battery box (4) wheel motor (5), disk motor (6) and the Electronics box (7).

it as light as possible. The disk is rotated by a brushed DC motor (6 in Figure 3.1) that is attached to the body. The body (2 in Figure 3.1) is made out of the same 1.5 $[mm]$ aluminum as the reaction disk. It is 350 $[mm]$ long, 110 $[mm]$ wide and 55 $[mm]$ deep. The driving wheel (3 in Figure 3.1) is rotated by another brushed DC motor attached to the body (5 in Figure 3.1) and an extension shaft goes from the motor through the wheel to a bearing at the other side. Above the bearing, a box (4 in Figure 3.1) is mounted containing a battery used to power the motors. Furthermore, in order to position the center of mass in the center of the body, a stabilization weight of 0.19 $[kg]$ is placed in the same box. Finally, in order to enclose all electronics, another box is placed right above the driving wheel.

In order to achieve as accurate parameters for the mathematical model as possible, a 3D model of the unicycle is set up in Autodesk's Fusion360, see Figure 3.1. All components are carefully given its correct dimensions and weight in order to get the inertia matrices of each part. All the parameters of the constructed unicycle can be found in Appendix A.

3.2 Electrical Construction

The electrical system can be divided into two subparts, a signal system and a driving system powered by separate voltage sources, see Figure 3.2. The task of the signal system is to collect data from sensors in order to determine the state of the unicycle and give control signals to the driving system. It consists of a 9 DOF inertial measurement unit (IMU), two motor encoders, a microcontroller, a 3.7 $[V]$ LIPO battery, three switches as well as other small parts.

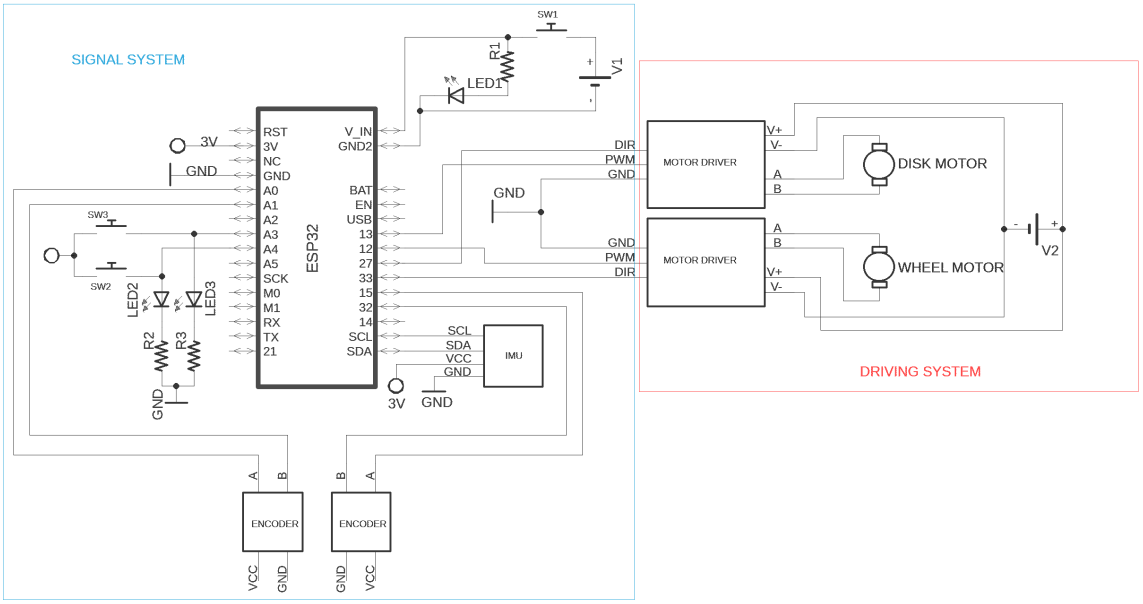


Figure 3.2: Schematics of the electrical construction, divided into one signal system and one driving system.

The IMU is STMicroelectronics LSM9DS1 (for the datasheet see [25]). It houses a 3-axis accelerometer, a 3-axis gyroscope and a 3-axis magnetometer, which will be used to determine the current angle and angular velocity of the unicycle (see Section 3.4.1).

The motor encoders are quadrature rotary encoders mounted on the back of the two DC motors. These encoders will be used to get the angular velocity of the driving wheel and the reaction disk (see Section 3.4.1).

The microcontroller is Adafruits dual core ESP32 Feather (for the datasheet see [26]). It has 520 [KB] SRAM and 4 [MB] flash which comes in handy for managing both the traditional and deep learning control algorithms. One core is dedicated to handle the control algorithms while the other takes care of the sensor readings and the user inputs. This is to make sure that the control algorithm is not interrupted by other tasks or external interrupts, which is described in detail in Section 3.4.1.

The switches are implemented in order to easily activate or deactivate different parts of the signal system. Switch SW1 is a switch between the microcontroller and the LIPO battery. Note that if a USB cable is plugged in, the signal system will be driven by the power from the USB regardless of the state of switch SW1. Switch SW2 and SW3 are used to activate the control system and switch between the two algorithms respectively.

The battery V1 in Figure 3.2 is the 3.7 $[V]$ LIPO battery and is used to drive the signal system. It can be charged by plugging in a USB cable to the microcontroller

and activating switch SW1. This is to avoid opening the box every time the battery needs to be charged.

The driving system consists of a four cell LIPO battery, two brushed DC motors and two motor drivers. Both motors are 12 $[V]$ brushed gear motors connected to separated motor drivers. The disk motor has a larger torque (2.16 $[Nm]$ stall) and speed (437 [RPM] at no load) than the driving wheel motor to be able to rotate the disk fast enough to generate sufficient torque to maintain stability. For the datasheets of the reaction disk motor and the wheel motor, see [27] and [28], respectively. The motor drivers are dimensioned to handle a stall current up to 30 $[A]$. For the datasheet of the motor drivers, see [29].

The two motor drivers and the entire signal system, except for the motor encoders, are mounted inside the electronics box. The motor encoders are mounted on the back of each motor and are protected with motor caps.

The final result of the mechanical and electrical design can be seen in Figure 3.3.

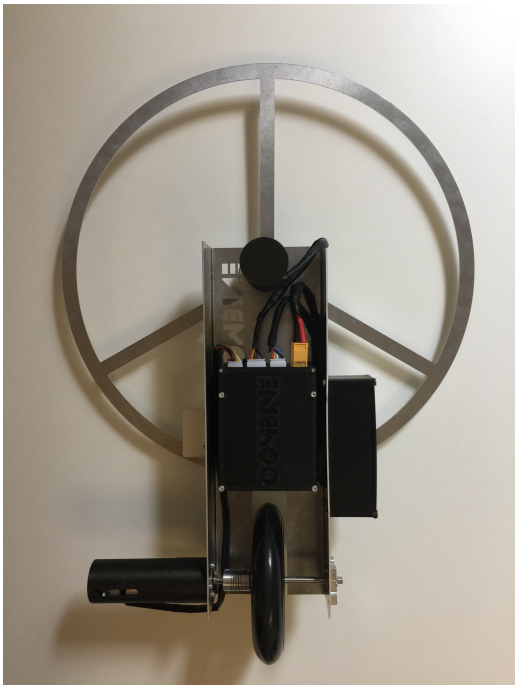


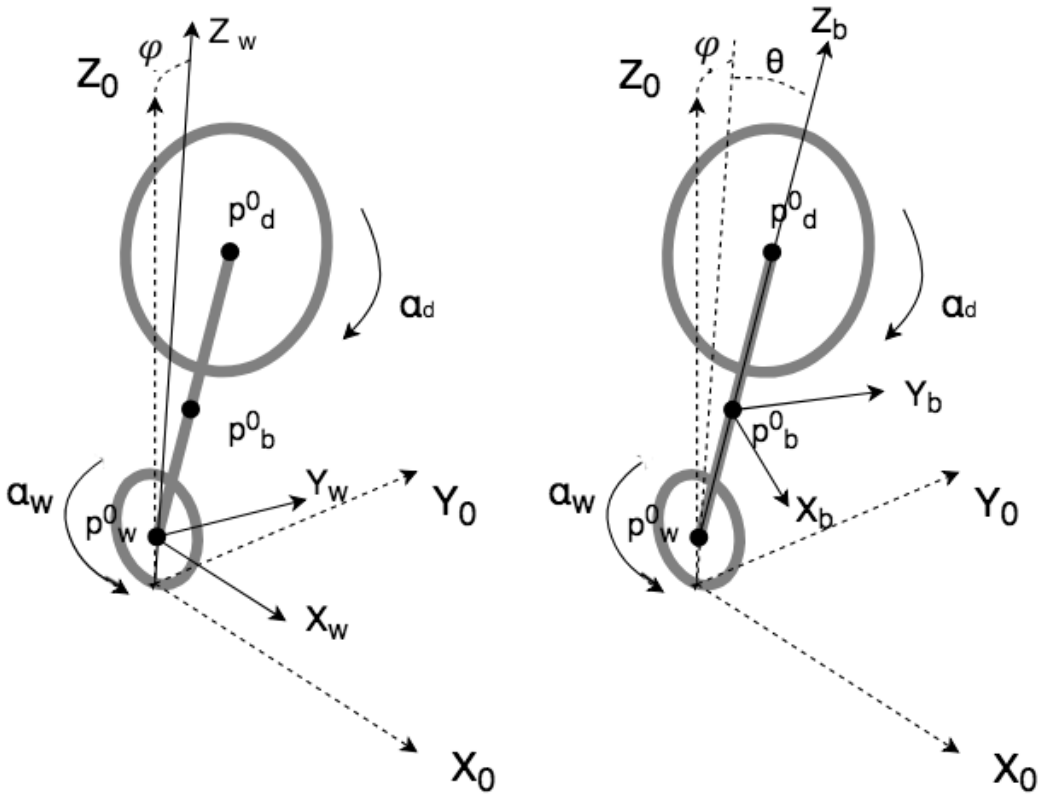
Figure 3.3: Overview of the result of the mechanical and electrical design.

3.3 Mathematical Model

Due to the systems multiple rotating frames, the Lagrangian method (see Section 2.2) is used for deriving the model. The unicycle is divided into 3 parts, the wheel, the body and the reaction disk. Four frames are then used to describe the system. The first frame is the world frame, which represents a space fixed coordinate

system independent of the system itself (see Figure 3.4). Its axes are denoted X_0, Y_0 and Z_0 and the direction of the driving wheel of the unicycle is assumed to be fixed along the X_0-axis giving the system its first DOF. The roll angle of the system is defined around the X_0-axis adding the second DOF to the system. The driving wheel has a separate frame (see Figure 3.4a), which will be referred to as the wheel frame. Note that the wheel frame is not affected by the spin of the wheel. Its axes are denoted X_w, Y_w and Z_w and the pitch of the system is defined as the rotation of the body and disk around the Y_w axis, giving the third DOF to the system. Furthermore, the body frame is located at the center of mass of the body and its axes are denoted X_b, Y_b and Z_b. Finally, the disk frame is located at the center of mass of the disk and its axes are denoted X_d, Y_d and Z_d. The rotation of the disk around X_d adds the final DOF to the system.

Thus the model of the unicycle has 4 degrees of freedom; the spin of the wheel (α_w), the roll of the system (φ), the pitch of the system (θ) and rotation of the disk (α_d). When referred to mutually, the roll and the pitch of the system will be specified as the system angles.



(a) The position of the wheel frame. **(b)** The position of the body frame.

Figure 3.4: An illustration of the system showing its angles and positions of frames used for the mathematical model.

3.3.1 Dynamical Parameterization

The position of the origin of the wheel frame can be expressed in the world frame according to

$$\boldsymbol{p}_{\boldsymbol{w}}^{\mathbf{0}} = \begin{bmatrix} -r_w \alpha_w \\ -r_w \sin(\varphi) \\ r_w \cos(\varphi) \end{bmatrix}, \tag{3.1}$$

where r_w is the radius of the wheel. Its value can be found in Appendix A.

The rotation from the wheel frame to the world frame R_w^0 and the rotation from the body frame to the wheel frame R_b^w are given by

$$R_w^0 = R_x^T \quad \text{and} \quad R_b^w = R_y^T \tag{3.2}$$

respectively, where R_x and R_y are the elementary rotations defined in Equation 2.4 in Section 2.1. The position of the origin of the body frame and the disk frame can, in the wheel frame, be expressed by

$$\boldsymbol{p}_{\boldsymbol{b}}^{\boldsymbol{w}} = R_b^w \begin{bmatrix} 0 \\ 0 \\ L_{wb} \end{bmatrix} \quad \text{and} \quad \boldsymbol{p}_{\boldsymbol{d}}^{\boldsymbol{w}} = R_b^w \begin{bmatrix} 0 \\ 0 \\ L_{wd} \end{bmatrix}. \tag{3.3}$$

respectively, where L_{wb} is the length from the center of the driving wheel to the center of mass of the body, while L_{wd} is the distance from the center of the driving wheel to the center of the disk (see Appendix A). In order to express the origin of the body frame and the disk frame in the world frame, one has to do the transformations

$$\boldsymbol{p}_{\boldsymbol{b}}^{\mathbf{0}} = \boldsymbol{p}_{\boldsymbol{w}}^{\mathbf{0}} + R_w^0 \boldsymbol{p}_{\boldsymbol{b}}^{\boldsymbol{w}} \quad \text{and} \quad \boldsymbol{p}_{\boldsymbol{d}}^{\mathbf{0}} = \boldsymbol{p}_{\boldsymbol{w}}^{\mathbf{0}} + R_w^0 \boldsymbol{p}_{\boldsymbol{d}}^{\boldsymbol{w}}, \tag{3.4}$$

which results in

$$\boldsymbol{p}_{\boldsymbol{b}}^{\mathbf{0}} = \begin{bmatrix} -\alpha_w r_w + L_{wb} \sin(\theta) \\ -\sin(\varphi)(r_w + L_{wb} \cos(\theta)) \\ \cos(\varphi)(r_w + L_{wb} \cos(\theta)) \end{bmatrix} \quad \text{and} \quad \boldsymbol{p}_{\boldsymbol{d}}^{\mathbf{0}} = \begin{bmatrix} -\alpha_w r_w + L_{wd} \sin(\theta) \\ -\sin(\varphi)(r_w + L_{wd} \cos(\theta)) \\ \cos(\varphi)(r_w + L_{wd} \cos(\theta)) \end{bmatrix}. \tag{3.5}$$

3.3.2 Lagrange's Equations

Recall, from Section 2.2, that a set of n coordinates, where n equals the DOF, which can uniquely describe the position of a system are called free generalized coordinates. Aiming to obtain free generalized coordinates, the trivial choice of coordinates $\boldsymbol{q}$ for the unicycle are

$$\boldsymbol{q}(t) = \begin{bmatrix} \alpha_w(t) \\ \alpha_d(t) \\ \varphi(t) \\ \theta(t) \end{bmatrix}. \tag{3.6}$$

Furthermore, recall that the Lagrangian of a system is defined according to

$$\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = T(\boldsymbol{q}, \dot{\boldsymbol{q}}) - U(\boldsymbol{q}), \tag{3.7}$$

where T is the kinetic energy and U is the potential energy of the system. For the unicycle, the potential energy of the system is given by

$$U(\boldsymbol{q}) = g \begin{bmatrix} m_w & m_b & m_d \end{bmatrix} \begin{bmatrix} p_w^0 \\ p_b^0 \\ p_d^0 \end{bmatrix}, \tag{3.8}$$

where m_w, m_b and m_d are masses of wheel, body and disk respectively. Their values are specified in appendix A.

The kinetic energy can be divided into one translational part $T_t(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and one rotational part $T_r(\boldsymbol{q}, \dot{\boldsymbol{q}})$. The translational kinetic energy is given by

$$T_t(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2} m_w \dot{\boldsymbol{p}}_{\boldsymbol{w}}^{\boldsymbol{0}}(\boldsymbol{q}) \dot{\boldsymbol{p}}_{\boldsymbol{w}}^{\boldsymbol{0}}(\boldsymbol{q})^T + \frac{1}{2} m_b \dot{\boldsymbol{p}}_{\boldsymbol{b}}^{\boldsymbol{0}}(\boldsymbol{q}) \dot{\boldsymbol{p}}_{\boldsymbol{b}}^{\boldsymbol{0}}(\boldsymbol{q})^T + \frac{1}{2} m_d \dot{\boldsymbol{p}}_{\boldsymbol{d}}^{\boldsymbol{0}}(\boldsymbol{q}) \dot{\boldsymbol{p}}_{\boldsymbol{d}}^{\boldsymbol{0}}(\boldsymbol{q})^T, \tag{3.9}$$

where $\dot{\boldsymbol{p}}_{\boldsymbol{i}}^{\boldsymbol{0}}(\boldsymbol{q}) = \frac{\partial \boldsymbol{p}_{\boldsymbol{i}}^{\boldsymbol{0}}}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}}$ is the time derivative of $\boldsymbol{p}_{\boldsymbol{i}}^{\boldsymbol{0}}$.

The rotational kinetic energy of the unicycle is derived for each part of the unicycle. To begin with, there are two rotations acting on the driving wheel. The first rotation is the spin of the driving wheel caused by the motor and the second is the roll (φ) around the X_0 axis. Hence, the total rotational velocity of the wheel expressed in the wheel frame is given by

$$\boldsymbol{\omega}_{\boldsymbol{w}}^{\boldsymbol{w}} = R_x \begin{bmatrix} \dot{\varphi} \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ \dot{\alpha}_w \\ 0 \end{bmatrix} = \begin{bmatrix} \dot{\varphi} \\ -\dot{\alpha}_w \\ 0 \end{bmatrix}. \tag{3.10}$$

The rotational kinetic energy of the wheel can thus be defined as

$$T_w = \frac{1}{2} \boldsymbol{\omega}_{\boldsymbol{w}}^{\boldsymbol{w}T} I_w \boldsymbol{\omega}_{\boldsymbol{w}}^{\boldsymbol{w}}, \tag{3.11}$$

where I_w is the inertia of the driving wheel expressed in the wheel frame (see Appendix A.1).

The angular velocity of the body in the body frame is given by

$$\boldsymbol{\omega}_{\boldsymbol{b}}^{\boldsymbol{b}} = R_y R_x \begin{bmatrix} \dot{\varphi} \\ 0 \\ 0 \end{bmatrix} + R_y \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} = \begin{bmatrix} \dot{\varphi} \cos(\theta) \\ \dot{\theta} \\ \dot{\varphi} \sin(\theta) \end{bmatrix}. \tag{3.12}$$

The total rotational kinetic energy of the body expressed in the body frame is thus given by

$$T_b = \frac{1}{2} \boldsymbol{\omega}_{\boldsymbol{b}}^{\boldsymbol{b}} I_b \boldsymbol{\omega}_{\boldsymbol{b}}^{\boldsymbol{b}}, \tag{3.13}$$

where I_b is the inertia of the body around its center of mass (see Appendix A.2). In the body, both motors (5 and 6 in Figure 3.1), the body itself (2 in Figure 3.1), the electronics (7 in Figure 3.1), the battery and the stabilization weight (4 in Figure 3.1) are included.

At last, the rotational velocity of the disk expressed in the body frame is given by

$$\boldsymbol{\omega_d^b} = \boldsymbol{\omega_b^b} + \begin{bmatrix} \dot{\alpha_d} \\ 0 \\ 0 \end{bmatrix}, \tag{3.14}$$

where $\dot{\alpha}_d$ is the rotation of the disk around the X_b-axis due to the motor torque. The rotational kinetic energy of the disk is given by

$$T_d = \frac{1}{2}\boldsymbol{\omega_d^b}^T I_d \boldsymbol{\omega_d^b}, \tag{3.15}$$

where I_d is the inertia of the disk taken around the center of the disk (see Appendix A.3).

Finally, the total rotational kinetic energy is given by

$$T_r(\boldsymbol{q}, \dot{\boldsymbol{q}}) = T_w + T_b + T_d. \tag{3.16}$$

Using Equation 3.9 and 3.16, the total kinetic energy is given by

$$T(\boldsymbol{q}, \dot{\boldsymbol{q}}) = T_t(\boldsymbol{q}, \dot{\boldsymbol{q}}) + T_r(\boldsymbol{q}, \dot{\boldsymbol{q}}). \tag{3.17}$$

Recall that the equation of motions is given by Equation 2.11. Inserting Equation 3.7, one gets

$$\frac{\partial}{\partial \dot{\boldsymbol{q}}}\left(\frac{\partial T(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \dot{\boldsymbol{q}}}\right)\ddot{\boldsymbol{q}} + \frac{\partial}{\partial \boldsymbol{q}}\left(\frac{\partial T(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \dot{\boldsymbol{q}}}\right)\dot{\boldsymbol{q}} - \frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \boldsymbol{q}} = \boldsymbol{Q}. \tag{3.18}$$

3.3.3 External Forces

The external forces on the system come from the disk and wheel motors. The torque generated from the motors should be expressed in the generalized coordinates (see Equation 2.9).

A positive torque from the wheel motor will result in a direct positive displacement in α_w as well as in θ. Hence, Q_1 and Q_4 are given by

$$Q_1 = \tau_w \quad \text{and} \quad Q_4 = \tau_w. \tag{3.19}$$

Similarly, a positive torque from the disk motor will result in a direct positive displacement in α_d and a direct negative displacement in φ. Hence, Q_2 and Q_3 are given by

$$Q_2 = \tau_d \quad \text{and} \quad Q_3 = -\tau_d. \tag{3.20}$$

The total external generalized forces are given by

$$\boldsymbol{Q} = \begin{bmatrix} \tau_w \\ \tau_d \\ -\tau_d \\ \tau_w \end{bmatrix}. \tag{3.21}$$

According to Kirchoffs voltage law, the dynamics of a DC motor can be described by

$$-u(t) + R_a i(t) + L_a \frac{d}{dt} i(t) + u_m(t) = 0, \tag{3.22}$$

where u is the input voltage, R_a is the armature resistance, L_a is the armature inductance, i is the armature current, $u_m(t) = K_u \omega_m(t)$ is the back emf voltage, $T_m(t) = K_m i(t)$ is the torque generated at the motor shaft, K_u is the motor velocity constant and K_m is the motor torque constant. After measurement, the inductance of both motors are negligible, and Equation 3.22 simplifies to

$$-u(t) + R_a i(t) + u_m(t) = 0. \tag{3.23}$$

The total external generalized forces from Equation 3.21 can now be written according to

$$\boldsymbol{Q} = \begin{bmatrix} K_{m,w} i_w(t) \\ K_{m,d} i_d(t) \\ -K_{m,d} i_d(t) \\ K_{m,w} i_w(t) \end{bmatrix}. \tag{3.24}$$

By combining Equation 3.24 and 3.23, one gets

$$\boldsymbol{Q}(\dot{\boldsymbol{q}}, \boldsymbol{u}) = \begin{bmatrix} K_{m,w} \frac{u_w(t) - K_{u,w} \dot{\alpha}_w(t)}{R_{a,w}} \\ K_{m,d} \frac{u_d(t) - K_{u,d} \dot{\alpha}_d(t)}{R_{a,d}} \\ -K_{m,d} \frac{u_d(t) - K_{u,d} \dot{\alpha}_d(t)}{R_{a,d}} \\ K_{m,w} \frac{u_w(t) - K_{u,w} \dot{\alpha}_w(t)}{R_{a,w}} \end{bmatrix}, \tag{3.25}$$

where $R_{a,w}$, $R_{a,d}$, $K_{m,w}$, $K_{m,d}$, $K_{u,w}$ and $K_{u,d}$ can be found in Appendix A.4. Finally, the dynamics of the system can be described by

$$\frac{\partial}{\partial \dot{\boldsymbol{q}}} \left(\frac{\partial T(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \dot{\boldsymbol{q}}} \right) \ddot{\boldsymbol{q}} + \frac{\partial}{\partial \boldsymbol{q}} \left(\frac{\partial T(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \dot{\boldsymbol{q}}} \right) \dot{\boldsymbol{q}} - \frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \boldsymbol{q}} - \boldsymbol{Q}(\dot{\boldsymbol{q}}, \boldsymbol{u}) = \boldsymbol{0}, \tag{3.26}$$

where $\boldsymbol{u} = \begin{bmatrix} u_w & u_d \end{bmatrix}^T$. The MATLAB implementation of the nonlinear model describing the dynamics of the system can be found in the open source repository, see section 1.5.

3.3.4 Simulation

In order to simulate the continuous time nonlinear model, an integration method needs to be implemented. One of the most widely used methods is the Runge Kutta 4 (RK4) method due to its simplicity and good stability characteristics [30]. However, due to stiffness in the model, the RK4 method fails to accurately simulate the model. A solution to this problem is implementing a variable step size solver that will automatically decrease the step size for steep gradients. The order 4/5 Fehlberg solver (RK45) is a common variable step size solver and yields a satisfactory result in simulations [30]. The implementation of the solver can be found in the open source repository, see section 1.5.

3.4 System Software Design

The developed system software design which can be found in the open source repository (see Section 1.5) is described in this section. It mainly consists of two tasks which are executed on one core each (see Figure 3.5). The first task is the State update task, which assigns the state, given the sensor readings. To do this, it utilizes the two classes Motor and IMU. The filtering and sensor fusion of the states are done in the corresponding classes.

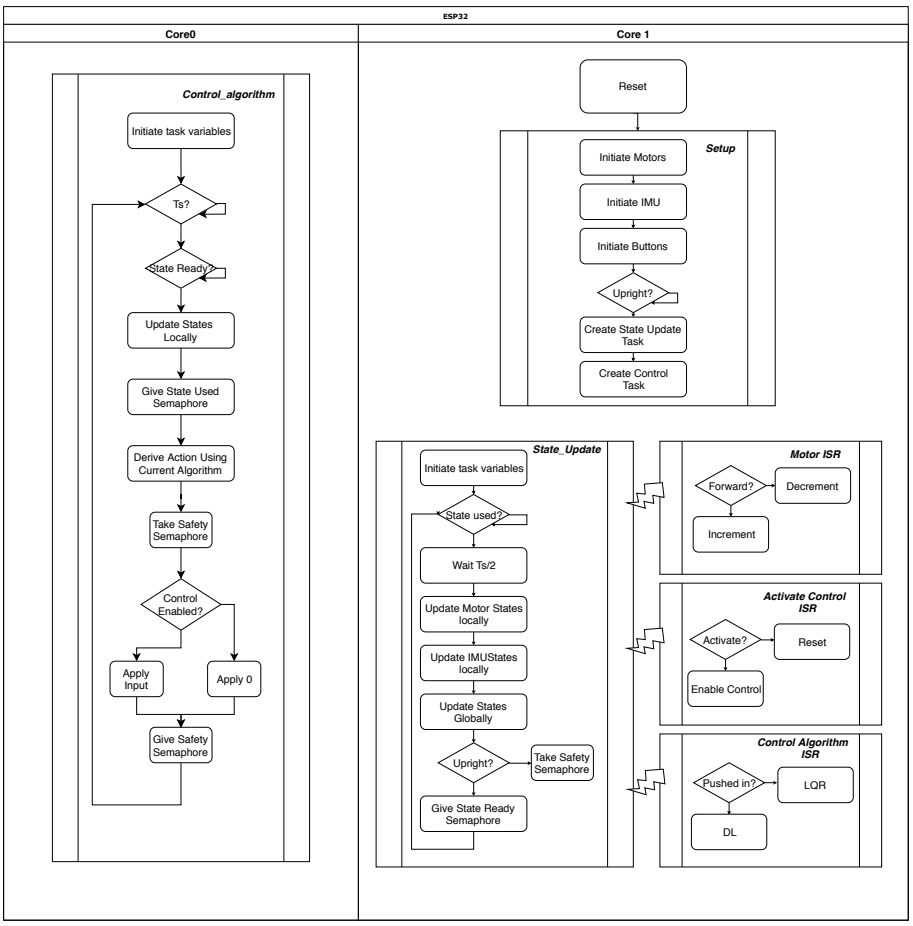


Figure 3.5: A simplified flowchart for the implemented software algorithm.

The second task is the Control algorithm task which, given the state of the external switches will execute either the traditional or the DL control algorithm. Once the input is derived, the task will apply the voltage to the motors using the Motor class.

In order to avoid data races and to properly time the two tasks, semaphores are used. One mutual exclusion semaphore protects the global state. Furthermore, there are two binary semaphores in order to properly time the Control algorithm task to execute after the states have been updated. The purpose of the semaphores is to execute the State update task before the next control update while having as fresh readings as possible. Finally, another semaphore is used in order to shut down

the entire system in case of the unicycle falling down.

At reset, the algorithm will run a setup routine before any of the tasks begin to execute. This setup routine will initiate all objects, semaphores, interrupts and states. The IMU will, depending on the state of the unicycle, calibrate. The two tasks will not start to execute unless the unicycle is in an upright position.

3.4.1 Sensor Filtering

In this section, the filtering and sensor fusion of all measurements will be described.

Motor state

As mentioned previously, both motors are equipped with a gearbox and a rotary encoder. The rotary encoders are quadrature incremental encoders, i.e. there are two sensors placed with a 90 degree phase difference which generate pulses. Depending on which channel generates the pulse first, one can not only determine the speed of the motor, but also the direction. Hence, by keeping track of the amount and order of the pulses, one will get the exact motor position and can estimate the velocity.

In the Motor class, an external interrupt is implemented which increments or decrements a counter depending on the state of the two sensor channels. The quadrature encoder of the disk motor has 12 counts per channel and revolution on the motor side, which corresponds to a counts to radians conversion of $\kappa_d = 0.0136$. The quadrature encoder of the wheel motor has 16 counts per channel and revolution on the motor side, which corresponds to a counts to radians conversion of $\kappa_w = 0.0073$. Using this, the angular velocity of the motor can be derived at each sample time by integrating the counter according to

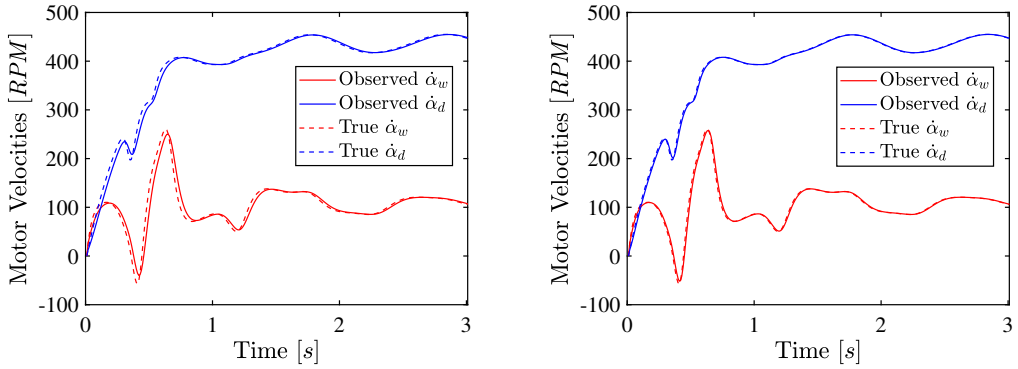
$$\omega_s = \frac{\Delta c \kappa}{T_s}, \tag{3.27}$$

where Δc is the number of counts since last update, κ is the conversion constant and T_s is the sample time. This results in a resolution of $r_w \approx 7$ [RPM] for the wheel motor and $r_d = 13$ [RPM] for the disk motor. Hence, some sort of a low pass filter should be implemented.

Recall the low pass filter with adaptive time constant given by Equation 2.30 in section 2.6. By using this filter with $\lambda = 0.7$, the worst (steady state) time constant is derived to

$$\tau_{motor} = wT_s = 0.0333. \tag{3.28}$$

A simulation using this tuning on both motor velocities, with a step input on both motors, is compared to the ideal simulation in Figure 3.6a. One can see that the dynamics are relatively slow and not sufficiently fast to neglect.



(a) $\lambda = 0.7$ **(b)** $\lambda = 0.45$

Figure 3.6: Simulation using a step input on both motors and filtered motor velocities with different tunings of the filter constant.

After trial and error, the constant was tuned to $\lambda = 0.45$ instead, in which case the noise of the motor velocities is considered acceptable, while the time constant ends up at

$$\tau_{motor} = wT_s = 0.0167. \tag{3.29}$$

A simulation using this tuning on the motor velocities with a step input on both motors is compared to the ideal simulation in Figure 3.6b. The dynamics of the filter is considered negligible.

System Angles

The roll angle φ and the pitch angle θ can be obtained using the accelerometer and the gyroscope in the IMU. By integrating the gyroscope data, one achieves an orientation estimation that is accurate on short term but drifts over time. However, the acceleration data is more accurate under the assumption that the IMU is not affected by any forces. This assumption can normally be made in a long term but when the unit actually moves, it is often exposed to external forces [16]. By fusing the two sensors using a complementary filter (see section 2.7), a more accurate estimation of the orientation can be made.

Assume that the unicycle is located in an arbitrary orientation as shown in Figure 3.7, where F_0 is the world coordinate frame, $F_{\text{IMU,B}}$ is the base frame of the IMU and F_{IMU} is the body frame of the IMU. Furthermore, it is assumed that the IMU is not exposed to any external forces. Then the acceleration readings of the IMU, expressed in the unit $\left[\frac{gm}{s^2}\right]$ can be expressed by

$$\begin{bmatrix} a_x^{\text{IMU}} \\ a_y^{\text{IMU}} \\ a_z^{\text{IMU}} \end{bmatrix} = R_0^{\text{IMU}} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = R_{\text{IMU,B}}^{\text{IMU}} R_0^{\text{IMU,B}} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{3.30}$$

Note that due to the assumption of no external forces, the IMU should be put as close to the center of rotation as possible. The rotation between the world frame

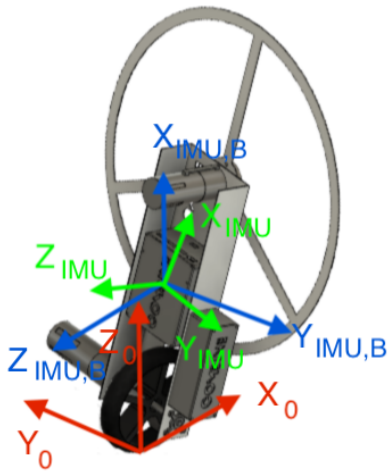


Figure 3.7: A simple illustration of the three frames used to approximate the system angles from the IMU readings.

and the IMU base frame is given by

$$R_0^{\text{IMU,B}} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{bmatrix}. \tag{3.31}$$

As the pitch rotation follows the roll rotation, $R_{\text{IMU,B}}^{\text{IMU}}$ is given by the yz rotation in the base frame of the IMU. Hence, Equation 3.30 can be written as

$$\begin{bmatrix} a_x^{\text{IMU}} \\ a_y^{\text{IMU}} \\ a_z^{\text{IMU}} \end{bmatrix} = R_{y\text{IMU,B}} R_{z\text{IMU,B}} R_0^{\text{IMU,B}} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \tag{3.32}$$

which evaluates to

$$\begin{bmatrix} a_x^{\text{IMU}} \\ a_y^{\text{IMU}} \\ a_z^{\text{IMU}} \end{bmatrix} = \begin{bmatrix} \cos(\theta)\cos(\varphi) \\ -\sin(\varphi) \\ \cos(\varphi)\sin(\theta) \end{bmatrix}. \tag{3.33}$$

By dividing the third row with the first row of Equation 3.33, one will get

$$\frac{a_z^{\text{IMU}}}{a_x^{\text{IMU}}} = -\frac{\sin(\theta)}{\cos(\theta)} = \tan(\theta). \tag{3.34}$$

Hence, using the accelerometer readings, the pitch angle at time step $t = kT_s$ can be estimated by

$$\hat{\theta}_{a,k} = \text{atan2}(\frac{a_{z,k}^{\text{IMU}}}{a_{x,k}^{\text{IMU}}}). \tag{3.35}$$

Dividing the second row of Equation 3.33 by $\cos(\varphi)$ results in

$$-\tan(\varphi) = \frac{a_y^{\text{IMU}}}{\cos(\varphi)}. \tag{3.36}$$

Furthermore, by adding the sum of the squares of the first and third row, one will get

$$(a_x^{\text{IMU}})^2 + (a_z^{\text{IMU}})^2 = \cos^2(\varphi)(\cos^2(\theta) + \sin^2(\theta)) = \cos^2(\varphi). \tag{3.37}$$

Taking the square root of Equation 3.37 results in

$$\cos(\varphi) = \sqrt{(a_x^{\text{IMU}})^2 + (a_z^{\text{IMU}})^2}. \tag{3.38}$$

Finally, combining Equation 3.36 and 3.38, one gets

$$-\tan(\varphi) = \frac{a_y^{\text{IMU}}}{\sqrt{(a_x^{\text{IMU}})^2 + (a_z^{\text{IMU}})^2}}. \tag{3.39}$$

Hence, using the accelerometer readings, the roll angle at time step $t = kT_s$ can be estimated by

$$\hat{\varphi}_{a,k} = -\text{atan2}\left(\frac{a_{y,k}^{\text{IMU}}}{\sqrt{(a_{x,k}^{\text{IMU}})^2 + (a_{z,k}^{\text{IMU}})^2}}\right). \tag{3.40}$$

$$\tag{3.41}$$

Using the gyroscope data, in order to get the estimation $\hat{\dot{\theta}}_g$ at time step $t = kT_s$, one needs to do the inverse transformation of the IMU base frame according to

$$\hat{\dot{\theta}}_{g,k} = -g_{y,k}^{\text{IMU}}, \tag{3.42}$$

where g_y^{IMU} is the gyroscope reading around the Y_{IMU} axis. Similarly, in order to get the estimation $\hat{\dot{\varphi}}_g$ one needs to do an inverse rotation in the IMU base frame and the inverse y rotation according to

$$\hat{\dot{\varphi}}_{g,k} = g_{x,k}^{\text{IMU}} \sin(\theta) - g_{z,k}^{\text{IMU}} \cos(\theta), \tag{3.43}$$

where g_z^{IMU} and g_x^{IMU} are the gyroscope readings around the Z_{IMU} and X_{IMU} axis respectively.

Finally, recall the complementary filter update step in Equation 2.32. Using Equation 3.35, 3.40, 3.42 and 3.43, the estimated system angles at time $t = kT_s$ are given by

$$\hat{\varphi}_k = (1 - \gamma_c)\hat{\varphi}_{a,k} + \gamma_c(\hat{\varphi}_{k-1} + T_s\dot{\varphi}_{g,k}) \tag{3.44}$$

$$\hat{\theta}_k = (1 - \gamma_c)\hat{\theta}_{a,k} + \gamma_c(\hat{\theta}_{k-1} + T_s\dot{\theta}_{g,k}), \tag{3.45}$$

where γ_c is a tuning parameter. By choosing $\gamma_c = 0.99$, a significant contribution from the gyroscope estimation is made in order to achieve fast dynamics of the filter while avoiding drift of the estimation.

As the acceleration state is not available directly in the achieved mathematical model, there is no trivial way of accurately simulating the complementary filter. However, when implemented in practice it yields satisfactory results.

Angular Velocities of the Unicycle

As mentioned previously, the angular velocities $\dot{\varphi}$ and $\dot{\theta}$ can be estimated as shown in Equations 3.42 and 3.43. Due to the noise on the sensor data, these states will be filtered using a Kalman filter as introduced in Section 2.8.

To begin with, a motion model of the angular velocity needs to be selected. Since the unicycle uses acceleration on the driving wheel and the reaction disk to stabilize, a random walk model is not considered sufficient. Hence, a constant velocity model of the gyroscope data (i.e. constant angular acceleration) will be used. As the angular acceleration is not measured, this state needs to be observed by the Kalman filter. Assuming white Gaussian noise, the motion and measurement model for both estimations at time step $t = kT_s$ are given by

$$\begin{bmatrix} \omega_k \\ \dot{\omega}_k \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix}}_{A} \begin{bmatrix} \omega_{k-1} \\ \dot{\omega}_{k-1} \end{bmatrix} + \begin{bmatrix} 0 \\ q_{k-1} \end{bmatrix} \tag{3.46}$$

$$y_{m,k} = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{H} \begin{bmatrix} \omega_k \\ \dot{\omega}_k \end{bmatrix} + r_k, \tag{3.47}$$

where $q_{k-1} \sim \mathcal{N}(0, \tilde{q})$, $r_k \sim \mathcal{N}(0, \tilde{r})$ and y_k is the sensor measurement. Using Monte Carlo approximation, the covariance of the noise on both sensor measurements are estimated as $\tilde{r}_{\dot{\varphi}} = \tilde{r}_{\dot{\theta}} = 5 \times 10^{-6}$. Furthermore, the model noise covariance is initially set to $\tilde{q}_{\dot{\varphi}} = \tilde{q}_{\dot{\theta}} = 5 \times 10^{-6}$.

The mean and covariance of both estimations are initialized to $\boldsymbol{m}_0 = \mathbf{0}_{2x1}$ and $P_0 = 0_{2\times 2}$. Due to this, the Kalman filters should be given some time to reach a steady state value of the covariance matrix P_k. This is done during the Setup routine.

Using the above defined parameters for both estimates $\hat{\dot{\varphi}}$ and $\hat{\dot{\theta}}$ in the update and prediction step defined in Equation 2.36 and 2.37, the amplitude of the noise is considered too large. After trial and error, the model noise covariance for both methods were tuned to $\tilde{q}_{\dot{\varphi}} = \tilde{q}_{\dot{\theta}} = 5 \times 10^{-6}$. This yields a satisfactory result both in practice and simulation. A simulation using two Kalman filters with a step input on the motors is shown in Figure 3.8. The dynamics of the Kalman filters are considered negligible.

3.5 Design Validation

There are many parameters that may contribute to an uncertainty in the model. Most of the mechanical parameters such as lengths, masses and inertias can be measured in reality or approximated in computer aided design (CAD) softwares. The parameters considered most critical are the motor constants from the two motors, as they are derived using other parameters from the corresponding data sheet (see Appendix A.4).

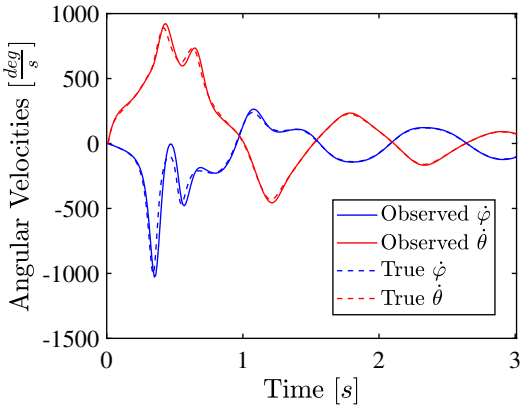
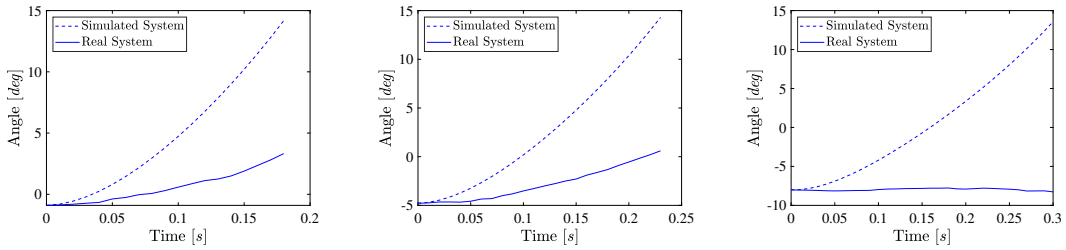


Figure 3.8: Simulation showing observed and true angular velocities $(\dot{\varphi}, \dot{\theta})$ of the system over time when using a step input on both motors. Observed states are shown as solid lines and true states are dashed.

In order to validate the accuracy of the model, the system step response is investigated both in simulation and practice. This is done by fixing the unicycle in a string to steadily stay in an initial state. Once the strings are cut, and the unicycle starts falling with an angular velocity ω_{th}, the step input is applied. Ideally, the threshold velocity should be infinitely small but due to practical uncertainties, it is set to $\omega_{th} = 0.0524\ [\frac{rad}{s}]$.

The first test is done by applying a constant input of $V_c = 10\ [V]$ on the disk motor. This is repeated around three different initial angles $\varphi_0 \approx -1\ [deg]$, $\varphi_0 \approx -5\ [deg]$ and $\varphi_0 \approx -8\ [deg]$. The data from all tests is recorded and simulations are performed from the same initial state that the step is applied. Using the nominal motor parameters, the results shown in Figure 3.9 are achieved.

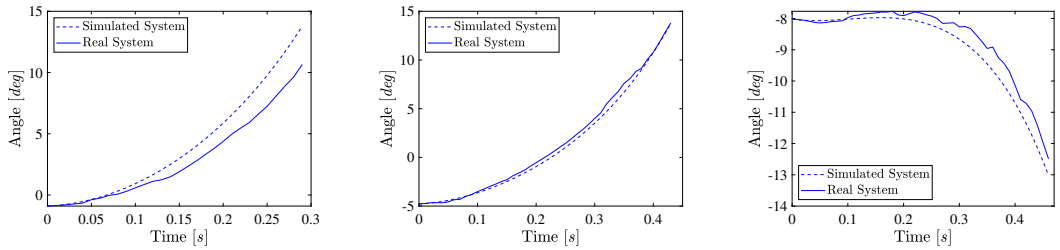


(a) $\varphi_0 = -0.9[deg]$ **(b)** $\varphi_0 = -4.8[deg]$ **(c)** $\varphi_0 = -8.0[deg]$

Figure 3.9: Comparison between test and simulation results using a step on the disk motor from three initial positions. Parameters used in simulation are the nominal parameters. The real test results are shown as solid lines and the simulation results are dashed.

From Figure 3.9a and 3.9b, a similar shape of the curve can be noted, however the torque generated from the motor seems to be too high. By tuning the disk motor parameter $K_{m,d}$ to $K_{m,d} = 0.0250$, the results shown in Figure 3.10 are achieved.

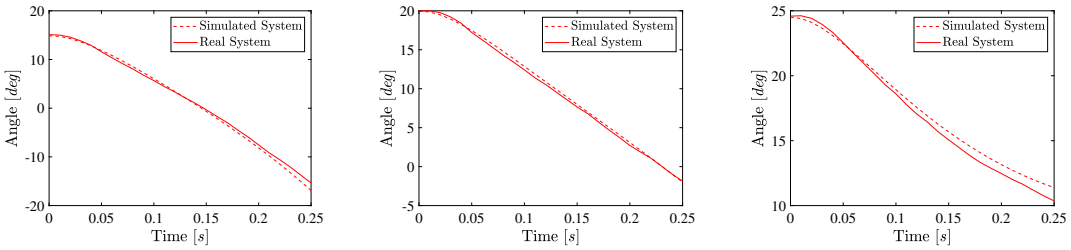
These simulations seem to better describe the real system.



(a) $\varphi_0 = -0.9\ [deg]$ **(b)** $\varphi_0 = -4.8\ [deg]$ **(c)** $\varphi_0 = -8.0\ [deg]$

Figure 3.10: Comparison between test and simulation results using a step on the disk motor from three initial positions. The simulation was made using the calibrated motor parameter. The real test results are shown as solid lines and the simulation results are dashed.

A similar step response test is made for the wheel motor in the initial angles $\theta_0 \approx 15\ [deg]$, $\theta_0 \approx 20\ [deg]$ and $\theta_0 \approx 25\ [deg]$. Using the nominal motor parameters of the wheel motor, the result shown in Figure 3.11 is achieved. Both the amplitude and the behaviour of the system seem to describe the system satisfactorily. More complex system identification is left for future work.



(a) $\theta_0 = 15.1\ [deg]$ **(b)** $\theta_0 = 20.0\ [deg]$ **(c)** $\theta_0 = 24.6\ [deg]$

Figure 3.11: Comparison between test and simulation results using a step on the wheel motor from three initial positions. The simulation was made using the nominal motor parameters. The real test results are shown as solid lines and the simulation results are dashed.

4

Control Strategies

In this chapter, the traditional and the DL algorithms will be implemented, both in simulation and in practice. In order to give a fair comparison between the two methods, they will be implemented under the same conditions. That means that the DL method will be trained using the same state information that is used in the traditional control method.

4.1 Traditional Control Method

In control theory, there are many ways to implement a controller. The by far most common controller design is the Proportional-integral-derivative (PID) controller [15]. This is most probably due to its simplicity and clear physical interpretation of each tuning constant. However, in multiple-input-multiple-output (MIMO) systems, the complexity of tuning the constants increases significantly [14].

Another common controller is the LQR controller introduced in Section 2.5. The LQR is one of the most commonly solved optimal control problems [15]. As a mathematical model of the system is available and due to its characteristics, the LQR controller is implemented in this project.

4.1.1 Linearization

In order to synthesise a LQR controller, a LTI system is needed. Hence, the mathematical model developed in Section 3.3 needs to be linearized. The state vector $\boldsymbol{x}$ is chosen as the time derivative of the general coordinates $\boldsymbol{q}$, the roll and the pitch of the system, according to

$$\boldsymbol{x} = \begin{bmatrix} \dot{\alpha}_w & \dot{\alpha}_d & \dot{\varphi} & \dot{\theta} & \varphi & \theta \end{bmatrix}^T. \tag{4.1}$$

In order to express the dynamics on the form in Equation 2.13, Equation 3.26 needs to be solved for $\ddot{\boldsymbol{q}}$. In matrix form, the dynamics can be rewritten as

$$A_{NL}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})\ddot{\boldsymbol{q}} - B_{NL}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{q}, \boldsymbol{u}) = 0. \tag{4.2}$$

As the determinant of A_{NL} is nonzero, the solution is unique and given by

$$\ddot{\boldsymbol{q}} = A_{NL}^{-1} B_{NL}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) = \boldsymbol{g}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}). \tag{4.3}$$

Using $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$, the dynamics can now be expressed according to

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}), \tag{4.4}$$

where $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ is given by

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} \boldsymbol{g}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \\ \dot{\varphi} \\ \dot{\theta} \end{bmatrix}. \tag{4.5}$$

By solving $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{0}$, one gets

$$\boldsymbol{x}_s = \boldsymbol{0}_{6\times 1} \quad \text{and} \quad \boldsymbol{u}_s = \boldsymbol{0}_{2\times 1}. \tag{4.6}$$

Utilizing Equation 4.5 and 4.6, the system matrices A and B defined by Equation 2.17 are derived and presented in Appendix B.1. Finally, the linearized dynamics are given by

$$\dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + B\boldsymbol{u}(t). \tag{4.7}$$

4.1.2 Discretization

As the controller will be implemented in a microcontroller, the mathematical system needs to be discretized before using it to derive the controller. More specifically, as the controller will yield a constant reference signal of the input voltage in every time step, the zero order hold discretization method described in Section 2.4 should be used. Using Equation 2.20, the sample time $T_s = 0.01$ and the continuous time system matrices A and B derived in the previous section, the discrete time system matrices A_d and B_d are derived and presented in Appendix B.2. Hence, the discrete time LTI system is described by

$$\dot{\boldsymbol{x}}_{k+1} = A_d\boldsymbol{x}_k + B_d\boldsymbol{u}_k. \tag{4.8}$$

4.1.3 Discrete Time System Analysis

The derived discrete time LTI system given by Equation 4.8 should be analyzed in order to validate the correctness of the model and identify system characteristics. To begin with, both the discrete time LTI system and the nonlinear continuous time system are simulated to fall from an arbitrary initial state set to

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0.2\frac{\pi}{180} & 0.2\frac{\pi}{180} \end{bmatrix}^T. \tag{4.9}$$

The result can be shown in Figure 4.1. As expected, one can see that the discrete time LTI system behaves in a similar fashion as the continuous time nonlinear system for small angle deviations.

An n-dimensional discrete time LTI system is asymptotically stable if all eigenvalues are strictly within the unit circle in the complex plane [31]. In this case, the eigenvalues $\boldsymbol{\lambda}_e$ are derived to

$$\boldsymbol{\lambda}_e = \begin{bmatrix} 0.7454 & 1.0655 & 1.0590 & 0.8797 & 0.9549 & 0.9460 \end{bmatrix}^T, \tag{4.10}$$

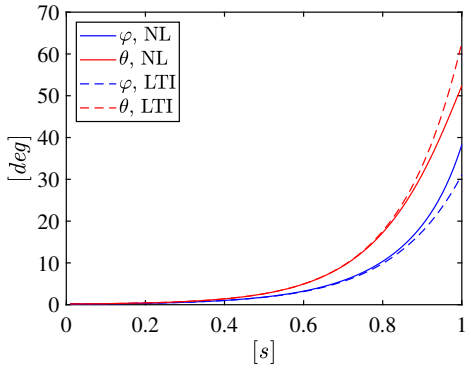


Figure 4.1: Simulation of the continuous time nonlinear system (solid) and the discrete time LTI system (dashed) from a small initial angle.

which implies that the system is unstable. This is of course expected due to the unstable linearization point.

Furthermore, an n-dimensional discrete time LTI system is controllable if the column vectors of the controllability matrix

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix} \tag{4.11}$$

span the n dimensional space [31]. Using the derived discrete LTI system, the controllability matrix $\mathcal{C}$ is derived and its rank is computed to

$$\text{rank}(\mathcal{C}) = 6, \tag{4.12}$$

which is equal the dimension of the system. As a numerical solver was used to derive the controllability matrix, the conditional number $\gamma_c \approx 4$ of the system is derived to ensure that it is well conditioned. As the conditional number is relatively low, one can draw the conclusion that the column vectors of the controllability matrix span the dimensional space and the discrete time LTI system is controllable.

4.1.4 Controller Synthesis

Recall from Section 2.5 that given a controllable discrete time LTI model and positive, symmetric and semi definite weight matrices $Q_x \geq 0$ and $Q_u > 0$, an optimal feedback law can be derived. By weighting the states and inputs equally according to

$$Q_x = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad Q_u = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{4.13}$$

the optimal steady state feedback gain matrix becomes

$$K = \begin{bmatrix} -2.3749 & -0.0003 & -0.0034 & 8.6670 & -0.0273 & 53.3768 \\ 0.0004 & -1.0748 & -33.2082 & -0.0070 & -189.2764 & -0.0224 \end{bmatrix}. \quad (4.14)$$

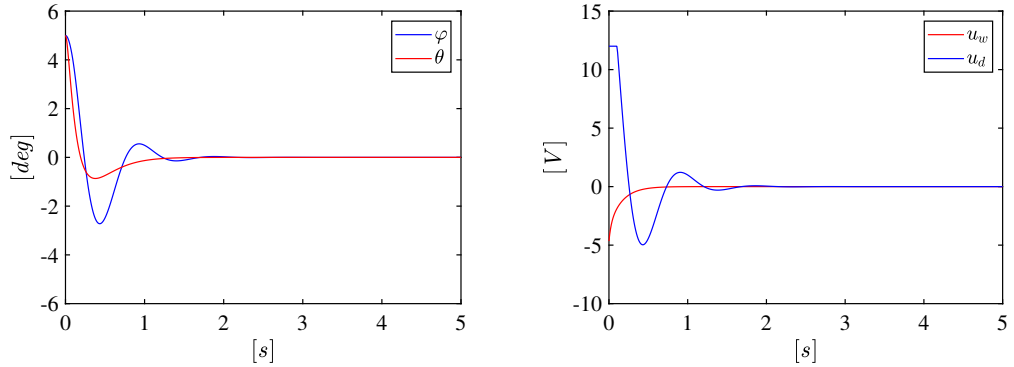
Using the optimal feedback law

$$\boldsymbol{u}_{opt,k} = -K\boldsymbol{x}_k, \quad (4.15)$$

a closed loop simulation of the nonlinear system from an initial state given by

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 5\frac{\pi}{180} & 5\frac{\pi}{180} \end{bmatrix}^T \quad (4.16)$$

is shown in Figure 4.2. One can see that the linear control law can stabilize the nonlinear model in a satisfactory way even with this trivial tuning.



(a) System angles **(b)** Motor voltage inputs

Figure 4.2: Closed loop simulation using the nonlinear continious time system model and the optimal control law achieved from trivial tuning and the original LTI system.

However, when implemented on the real system, the unicycle only manages to stay upright for a couple of seconds. The issue lies in an unstable, stuttering behaviour around the Y axis of the unicycle. All the states corresponding to that axis, i.e. the angular velocity of the wheel ($\dot{\alpha}_w$), the pitch velocity ($\dot{\theta}$) and the pitch angle (θ) was tuned as well as possible. However, no matter the tuning the issue was still present. After further investigation, the problem was found to most probably be due to an unmodelled backlash in the wheel.

To solve the backlash issue, one would like smoothen the inputs when the wheel changes its direction. A possible way to do this is to simply implement a filter on the input to the wheel. Recall Equation 2.29, in which a first order discrete IIR filter is described. Using the first order IIR filter, the input to the wheel motor is now given by

$$u_{w,k+1} = (1 - \frac{T_s}{\tau})u_{w,k} + \underbrace{\frac{T_s}{\tau}}_{\lambda_w} u_{f,k}, \quad (4.17)$$

where u_f is the input to the filter and $\lambda_w = 0.1$. Extending the discrete time LTI model with u_w as a state and u_f as a new input, gives

$$\begin{bmatrix} \boldsymbol{x}_{k+1} \\ u_{w,k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} A_d & \boldsymbol{b}_{d,1} \\ \mathbf{0}_{1\times 6} & 1-\lambda_w \end{bmatrix}}_{\tilde{A}_d} \underbrace{\begin{bmatrix} \boldsymbol{x}_k \\ u_{w,k} \end{bmatrix}}_{\tilde{\boldsymbol{x}}} + \underbrace{\begin{bmatrix} \mathbf{0}_{6\times 1} & \boldsymbol{b}_{d,2} \\ \lambda_w & 0 \end{bmatrix}}_{\tilde{B}_d} \underbrace{\begin{bmatrix} u_{f,k} \\ u_{d,k} \end{bmatrix}}_{\tilde{\boldsymbol{u}}}, \tag{4.18}$$

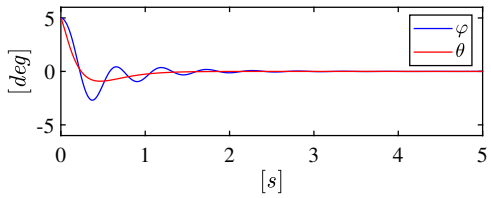
where $\boldsymbol{b_{d,1}}$ and $\boldsymbol{b_{d,2}}$ are the first and second columns of B_d. The result of the extended system matrices $\tilde{A}_d$ and $\tilde{B}_d$ can be found in Appendix B.3. Deriving the rank of the controllability matrix $\tilde{\mathcal{C}}$ of the extended system, gives

$$\operatorname{rank}(\tilde{\mathcal{C}}) = 7 \tag{4.19}$$

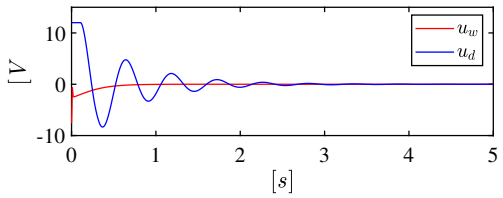
and the conditional number of the new system remains low. This implies controllability of the extended system. In order to avoid adding an extra weight on the wheel input, the additional state is given the weight $q_{u_w} = 0$ in the weight matrix Q_x and all other weights are kept unchanged. This results in an optimal steady state feedback matrix given by

$$K = \begin{bmatrix} -4.0731 & -0.0003 & -0.0039 & 13.8771 & -0.0356 & 87.4214 & 1.5365 \\ 0.0002 & -1.1972 & -51.1834 & -0.0091 & -293.6850 & -0.0237 & 0.0003 \end{bmatrix}. \tag{4.20}$$

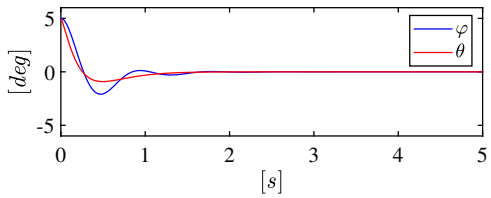
From the same initial state as in Equation 4.16, a simulation using the extended feedback gain is shown in Figures 4.3a - 4.3b. One can conclude that the extended



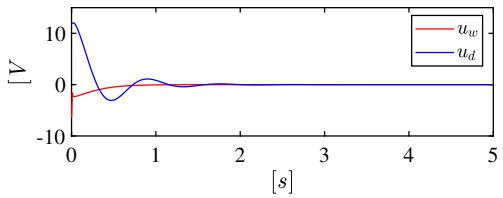
(a) Angles of the unicycle using controller achieved from trivial tuning.



(b) Motor voltage inputs using controller achieved from trivial tuning.



(c) Angles of the unicycle using final controller.



(d) Motor voltage inputs using final controller.

Figure 4.3: Closed loop simulation using the nonlinear continuous time system model and two versions of the control law computed with the extended system.

controller successfully stabilizes the nonlinear system with an undesirable oscillation. This can be resolved by tuning the weight matrices Q_x and Q_u. After trial and error,

the final weight matrices are given by

$$Q_x = \begin{bmatrix} 0.001 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.001 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad Q_u = 12 \cdot \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix} \tag{4.21}$$

which yields the final optimal steady state feedback gain matrix

$$K = \begin{bmatrix} -3.4573 & -0.0000 & 0.0007 & 11.6581 & -0.0073 & 73.8567 & 1.2872 \\ -0.0000 & -0.5240 & -27.4457 & -0.0045 & -158.8489 & -0.0102 & 0.0002 \end{bmatrix}. \tag{4.22}$$

As can be seen in Figures 4.3c and 4.3d, the final controller got smaller overshoot and less oscillation than the previous controller. Furthermore, once tested on the real system, it stabilizes with a satisfactory behaviour as well.

4.2 Deep Learning Algorithm

For the deep learning control of the unicycle, a Proximal Policy Optimization (PPO) is chosen, see Section 2.9.5. The method is built on a policy based reinforcement learning which offers practical ways of dealing with continuous spaces and infinite number of actions. Instead of learning probability for every action, policy based methods learn the distribution of probabilities over actions [18]. The official paper on PPO method [23] presents results from training the algorithm along with other ones on various applications. The PPO showed a superiority in complex control tasks compared to other policy based algorithms. It is as well considered to be the state-of-the-art method for reinforced learning in continuous spaces and is therefore chosen for the task [32].

4.2.1 DL Framework

An implementation of the PPO algorithm from OpenAI ™'s Baselines is used. In order to train an agent with the PPO method a special training environment needs to be set up using the Gym library. Gym is a platform aimed to standardise more environments used in publications in order to make it easier to reproduce public research and compare results from independent papers. In the environment, the agent can explore the properties of the physical system through simulation. Hence, the nonlinear mathematical model developed in Section 3.3 is implemented in Python to be used during training. Furthermore, in order to accurately simulate the continuous time nonlinear model, a RK45 solver is implemented, see Section 3.3.4. All Gym environments must have a common interface of a special environment class, consisting of the two base functions, Step and Reset.

The Step function observes the current state and uses the nonlinear model and Runge Kutta numerical integration to take a step with given inputs. Furthermore, the step function returns whether the resulting state is terminal or not, the reward and an observation of the new state.

The purpose of the Reset function is to set the initial states of the agent in the environment. This way, the agent can start from a new initial state every episode and explore the system. Hence, the reset function is called every time a new episode should be initiated.

The unicycle environment focuses on episodic setting of reinforced learning. The reward returned from the Step function is determined by a reward function and is used as a feedback to the agent. If the input which the agent selects leads to a terminal state, the episode will end and the agent will call the Reset function. Each step taken gives a reward and the agent's goal is to maximize that reward. In general, the more steps the agent manages to take without falling down, the more reward it will get for that episode. By repeating this procedure in the unicycle environment the agent generates its own training data and uses it to improve its policy.

4.2.2 Parameters

Learning Rate

An important tuning parameter during training is the learning rate introduced in Section 2.9.1. If the learning rate is selected too large the gradient will most likely fail to converge or even diverge and enter a region of the parameter space far away from the optimum. This can be avoided by reducing the learning rate. However, with a too small learning rate gradient decent can be slow and the training could end up without any results at all. In general, selecting a learning rate is a trade off between the larger learning rates fast convergence and the small learning rates long training.

To select the learning rate the previously mentioned paper on PPO will be used as a benchmark [23]. There, a learning rate of 3×10^{-4} was used to train an agent in a fairly complex continuous control problem for a million time steps. By monitoring the episodic reward during training, an appropriate learning rate should be selected. During early stages of the training, the learning rate α_r is set to $\alpha_r = 10^{-3}$. The learning rate will then be reduced when the model is fine tuned.

Observation and Action Spaces

All possible actions the agent can take in the environment are defined by the action space. The action space is determined by the maximum and minimum voltage to the motors and is therefore defined as

$$u_w \in \left[-\bar{u}_w, \quad \bar{u}_w\right] \quad u_d \in \left[-\bar{u}_d, \quad \bar{u}_d\right], \tag{4.23}$$

where $\bar{u}_w = \bar{u}_d = 12\ [V]$. Similarly, all possible states the agent's actions can lead to are defined by the observation space. If the system deviates too far away from the upright position, it will be practically impossible to stabilize. In order to prevent the agent from exploring regions too far from those that are stabilizable, the observation space is limited by terminal states. Thus, the observation space $\bar{\boldsymbol{x}}$ is defined as the state $\boldsymbol{x}$ with the limits

$$\dot{\alpha}_w \in \left[-\bar{\dot{\alpha}}_w, \quad \bar{\dot{\alpha}}_w\right], \qquad \dot{\alpha}_d \in \left[-\bar{\dot{\alpha}}_d, \quad \bar{\dot{\alpha}}_d\right], \qquad \dot{\varphi} \in \left[-\bar{\dot{\varphi}} \quad , \bar{\dot{\varphi}}\right], \tag{4.24}$$

$$\dot{\theta} \in \left[-\bar{\dot{\theta}} \quad , \bar{\dot{\theta}}\right], \qquad \varphi \in \left[-\bar{\varphi} \quad , \bar{\varphi}\right], \qquad \theta \in \left[-\bar{\theta} \quad , \bar{\theta}\right], \tag{4.25}$$

where $\bar{\dot{\alpha}}_w = 84$ [RPM], $\bar{\dot{\alpha}}_d = 450$ [RPM], $\bar{\dot{\varphi}} = \bar{\dot{\theta}} = 100\ [\frac{deg}{s}]$ and $\bar{\varphi} = \bar{\theta} = 25\ [deg]$. If the agent finds itself in a state outside of the observation space (in a terminal state) the episode will end and the Reset function is called.

Reward

Selecting a suitable reward function is an essential factor when training a model. The reward is the feedback the agent receives from the environment. It is important to select a reward which reflects what should be accomplished. If the reward function is badly engineered the agent might learn an unexpected way to make the environment deliver a reward with an undesirable behaviour.

Designing a reward function is in general left to an informal trial-and error search for functions that produce acceptable results [18]. Using inspiration from other previous and similar projects, several reward functions are examined with various results. Below, some examples of the behaviour in the simulated nonlinear model at early stages of the training, using different reward functions, are presented.

A common reward function is to give 1 as a reward for an acceptable step in the environment and -1 for a step that leads to a system failure. Based on this, the reward function

$$R_k = \begin{cases} 1 & \text{for } \boldsymbol{x}_t \in \bar{\boldsymbol{x}} \\ -1 & \text{otherwise} \end{cases} \tag{4.26}$$

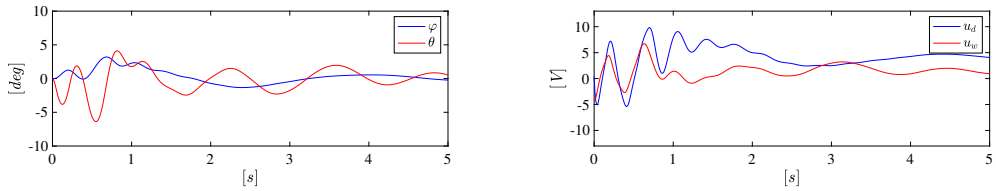
is implemented, where R_k is the reward at time step $t = kT_s$.
This leads to an upright but oscillating behaviour of the unicycle as can be seen in Figure 4.4. The system manages to maintain within the observation space, however, without the knowledge of the true goal of being close to zero. Another reward function is constructed with an increasing reward as θ and φ approaches zero, according to

$$R_k = (1 - \left(\frac{\varphi_k}{\bar{\varphi}}\right)^2) + (1 - \left(\frac{\theta_k}{\bar{\theta}}\right)^2), \tag{4.27}$$

where φ_k and θ_k is the roll and the pitch angle respectively at time instant $t = kT_s$. It is inspired by a function used in a continuous control problem of the Mountain car from OpenAI's Gym. The performance achieved from using this reward function can be seen in Figure 4.5. One can note that the reward function seems to have a reasonably good performance in the angles, while the input oscillates in an undesirable fashion. This is a typical example of a fairly simple reward function working well if the system is going to be simulated, however, in practice it will most probably be problematic. Hence, achieving a more complex and restricted behaviour with reasonable inputs for a real physical system calls for a more complex function.The final suggestion is a function constructed from all states according to

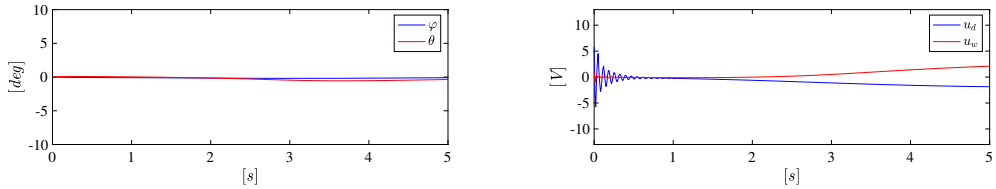
$$\sum_{i=1}^{6} \beta_i (1 - \left(\frac{x_{i,k}}{\bar{x}_i}\right)^2), \tag{4.28}$$

where $x_{i,k}$ is the current state, $\bar{x}_i$ is the maximum value for each state and β_i, $i = 1 \dots 6$ are scaling constants for each factor. Figure 4.6 shows a steady state performance gained from this reward function. In this example, $\beta_i = 1$ for all terms. As soon as the system ends up outside the observation space the episode is terminated and the agent receives a negative reward of -1.



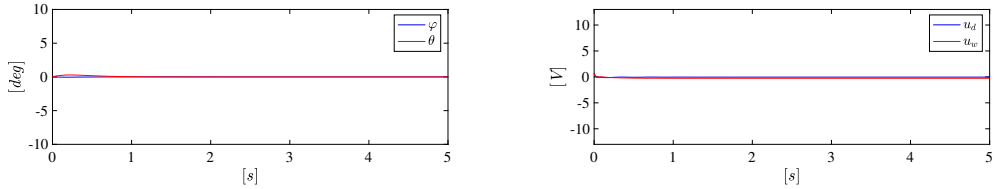
(a) System angles **(b)** Motor voltage inputs

Figure 4.4: An oscillating "stable" performance in simulation after million iterations using reward function from Equation 4.26.



(a) System angles **(b)** Motor voltage inputs

Figure 4.5: Simulation showing stable system angles after million iterations using reward function in Equation 4.27. The simulated motor voltage inputs however are not optimal.



(a) System angles **(b)** Motor voltage inputs

Figure 4.6: Stable performance of both system angles and motor inputs after million iterations using the reward function from Equation 4.28.

Reset Function

As previously mentioned, the purpose of the Reset function is to set the initial states of the agent in the environment. This is done every time a new episode starts. As a first attempt, the reset function selects an initial state according to

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & \varphi_0 & \theta_0 \end{bmatrix}^T, \tag{4.29}$$

where θ_0 and φ_0 are drawn according to a random uniform distribution from the observation space, i.e.

$$\varphi_0 \in \begin{bmatrix} -\bar{\varphi}, & \bar{\varphi} \end{bmatrix}, \qquad \theta_0 \in \begin{bmatrix} -\bar{\theta}, & \bar{\theta} \end{bmatrix}. \tag{4.30}$$

Using this reset function, the algorithm is only capable of stabilizing from a small initial angle. In order to force the policy to stabilize larger initial angles, the set from which θ_0 and φ_0 is selected is redefined to $\varphi_0 \in \Phi$ and $\theta \in \Theta$. The sets Φ and Θ are defined according to

$$\Phi = \begin{bmatrix} -\bar{\varphi}, & -\underline{\varphi} \end{bmatrix} \cup \begin{bmatrix} \underline{\varphi}, & \bar{\varphi} \end{bmatrix}, \qquad \Theta = \begin{bmatrix} -\bar{\theta}, & -\underline{\theta} \end{bmatrix} \cup \begin{bmatrix} \underline{\theta}, & \bar{\theta} \end{bmatrix}, \tag{4.31}$$

where $\bar{\varphi}$, $\underline{\varphi}$, $\bar{\theta}$ and $\underline{\theta}$ are increased the longer the agent has trained. This way, the algorithm first focuses to learn to stabilize smaller angles. Once it learns that, it can gradually learn to stabilize larger angles as well.

Finally, as the policy improves, the agent manages to make the system remain stable for a longer amount of time during training. Hence, it is therefore receiving a lot of reward for long episodes without exploring the system and improve its policy further. Due to this, a time limit is set for each episode. Once the episode exceeds $k = 15000$ steps, the reset function is called and the agent is therefore regularly forced to stabilize from a large angle during training.

4.2.3 Final ppo Model

The final model is first trained at a learning rate of $\alpha_r = 10^{-3}$ for $n_i = 10^7$ iterations. It is then fine tuned using a decreased learning of $\alpha_r = 10^{-6}$ for $n_i = 5 \times 10^5$ iterations at a time.

Furthermore, starting from $\bar{\varphi} = 7$ $[deg]$, $\underline{\varphi} = 3$ $[deg]$, $\bar{\theta} = 15$ $[deg]$ and $\underline{\theta} = 7$ $[deg]$, the upper bound of the reset limitations from Equation 4.31 are increased in 1 $[deg]$ at a time until there are no more improvement in the maximum initial angles the algorithm manages to stabilize. The final reset limitations are given by $\bar{\varphi} = 9$ $[deg]$, $\underline{\varphi} = 3$ $[deg]$, $\bar{\theta} = 22$ $[deg]$ and $\underline{\theta} = 7$ $[deg]$.

During the first $n_i = 10^6$ iterations, the scaling factors β_i from the reward function in Equation 4.28 are set to be 1 for all states. When fine tuning, β_2 is set to 1.5 in several iteration sets in order to try to reduce the aggressiveness of the disk in practice.

Transfer to Hardware

The final trained and fine tuned model is a mulitylayer precepton network with an input layer, two hidden layers and an output layer, see Section 2.9.2. The input layer and the hidden layers have a *tanh* activation function, see Section 2.9.3. The input layer has six nodes, one for each state. The hidden layers have 64 nodes each and the output layer has two final nodes. This setup can be interpreted as three cascaded matrix multiplications. The output $\boldsymbol{y_0}$ from the input layer is given by

$$\boldsymbol{y_0} = \tanh(W_0 \boldsymbol{x}_k + \boldsymbol{b}_0), \tag{4.32}$$

where W_0 and $\boldsymbol{b}_0$ are the weights and the biases of the input layer respectively. The output $\boldsymbol{y_1}$ from the first hidden layer is given by

$$\boldsymbol{y_1} = \tanh(W_1 \boldsymbol{y}_0 + \boldsymbol{b}_1), \tag{4.33}$$

where W_1 and $\boldsymbol{b}_1$ are the weights and the biases of the first hidden layer respectively. Similarly, the output $\boldsymbol{y_2}$ from the second hidden layer is given by

$$\boldsymbol{y_2} = \tanh(W_2 \boldsymbol{y}_1 + \boldsymbol{b}_2), \tag{4.34}$$

where W_2 and $\boldsymbol{b}_2$ are the weights and the biases of the second hidden layer respectively. Finally, the control law of the DL algorithm is given by

$$\boldsymbol{u}_k = W_3 \boldsymbol{y}_2 + \boldsymbol{b}_2, \tag{4.35}$$

where W_3 and $\boldsymbol{b}_3$ are the weights and the biases of the output layer, respectively.

When transferring the trained policy to the unicycle, it does not manage to stabilize. To solve this, noise was added to the observations in the simulation environment and additional fine tuning of the policy was done using the same training parameters. The added noise was set to a normal distributed Gaussian noise with zero mean and a standard deviation of 5×10^{-6}. When transferring the new policy to the real physical system it managed to stabilize in a satisfactory way.

5
Results

In this chapter, the final versions of the traditional LQR controller and the PPO control algorithm will be evaluated. The evaluation of the two methods will be done in simulation and in practice separately. In order to get a hint on the differences between simulation and the real system, the simulations will be evaluated using the final practical controllers given by Equation 4.22 and 4.35. Due to the fast dynamics of the sensor filters, see Section 3.4, they are not included in the simulations. However, the filter on the input signal u_w implemented due to the backlash in the ground wheel has considerably slower dynamics and is included in all simulations.

In order to achieve a viable definition of a stable state for the real system, a test of the steady state performance of the two control methods will be done in practice. To evaluate the performance and robustness of the methods, there will be 4 types of tests that will be executed in a similar fashion both in simulation and in practice. The intention of these tests is to evaluate the:

- Maximum initial angle,
- General performance metrics,
- Robustness to model errors,
- Robustness to external impulses.

The scope of the Maximum initial angle test is to evaluate what the maximal initial deviation from the zero state each control method can stabilize from. This way, one can get an idea of the interval of stabilization for both methods.

The General performance metrics tests are meant to evaluate the time of stabilization and the amount of overshoot of the methods. This should be done from a fixed initial state within interval of stabilization for both methods. The time of stabilization is defined as the time it takes to reach a stable state.

In order to get an idea of how the two methods perform when exposed to model errors, they will be evaluated with a slight translation of the center of mass. This will be tested in the Robustness to model errors test by evaluating the way each method stabilizes its extreme initial positions. Hence, the Maximum initial angle test will be repeated using the system with a translated center of mass.

Finally, the scope of the Robustness to external impulses test is to find the maximum external impulse each method can handle.

5.1 Test Procedure

In order to make the results replicable, all tests will be executed according to a predefined test procedure. The test procedure of all tests will be described in this section.

5.1.1 Steady State Performance Test

The Steady state performance test will be executed by starting the unicycle in a stable position and let it balance by itself for a time $t = 10$ $[s]$. The tests for the different methods will be executed several times consecutively in order to avoid any eventual temperature dependency from the IMU. The tests will be done in the same environment using the same calibration during all tests for both methods. The result from the tests should be the mean and standard deviation of the angle states as well as the motor states as these will be used as a metric to define the stable state.

5.1.2 Maximum initial angle

The maximum initial angle test will be executed in as close to perfect environment as possible. In simulation, this means simulating the two control methods using the non-linear model without any sensor or model noise. In practice, the unicycle will be evaluated with minimal external disturbances.

Suppose that the unicycle is to be tested from the initial state

$$\hat{\boldsymbol{x}}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & \hat{\varphi}_0 & \hat{\theta}_0 \end{bmatrix}^T. \tag{5.1}$$

In practice, the unicycle will be fixed in two strings before the test is initiated in order to get a steady initial position close to $\hat{\varphi}_0$ and $\hat{\theta}_0$. To avoid external interference on the system caused by the strings, the test will not be started until the strings are cut. Once the strings are cut, and the unicycle starts falling with a angular velocity ω_{th}, the control algorithm is activated. Ideally, the threshold velocity in which the control algorithm is activated should be infinitely small. However, due to practical uncertainties, it was set to $\omega_{th} = 3$ $[\frac{deg}{s}]$ after trial and error on the physical system. Hence, the initial state from which the test is performed will be

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 & 0 & \dot{\varphi}_0 & \dot{\theta}_0 & \varphi_0 & \theta_0 \end{bmatrix}^T, \tag{5.2}$$

where

$$|\dot{\varphi}_0| \in \begin{bmatrix} 0, & \omega_{th} + \varepsilon_1 \end{bmatrix} \qquad |\dot{\theta}_0| \in \begin{bmatrix} 0, & \omega_{th} + \varepsilon_2 \end{bmatrix} \tag{5.3}$$

$$\varphi_0 \in \begin{bmatrix} \hat{\varphi}_0 - \varepsilon_3, & \hat{\varphi}_0 + \varepsilon_3 \end{bmatrix} \qquad \theta_0 \in \begin{bmatrix} \hat{\theta}_0 - \varepsilon_4, & \hat{\theta}_0 + \varepsilon_4 \end{bmatrix} \tag{5.4}$$

and ε_i is a small number. Only tests in which the angle of the tested DOF and the corresponding angular velocity fulfills Equation 5.2 using $\varepsilon = 0.2$ $[deg]$ and $\varepsilon = 3$ $[\frac{deg}{s}]$ will be recorded. This procedure will be repeated until each DOF and method has

10 approved test recordings.

The maximum initial angle of a method in a specific DOF is regarded as the maximum integer in which the method stabilizes. In practice, a method is defined to stabilize a specific initial position given at least 5 out of the 10 approved tests have stabilized. This ratio was decided based on the fact that when performing the tests the human factor had a huge impact on the results due to the complexity of the execution of the tests.

5.1.3 General Performance Metrics

In order to compare time of stabilization and the overshoot of the two methods, the metrics should be evaluated from the same initial position using both methods. The test procedure described for the Maximum initial angle test will be repeated from an initial position within both methods interval of stabilization. In practice, each method should have 10 approved tests in each DOF, in which the time of stabilization and the overshoot is derived.

The time of stabilization is defined as the time when the tested initial angle and the corresponding motor velocity first is within the interval I_1 and I_2 defined by

$$I_1 = \begin{bmatrix} 0, & \delta_a \end{bmatrix} \qquad\qquad I_2 = \begin{bmatrix} 0, & \delta_\alpha \end{bmatrix}, \tag{5.5}$$

where δ_a and δ_α are chosen according to the Steady state performance test, see Section 5.2.

The General performance metrics test will be evaluated for a non zero initial position both in the X_0 and Y_0 axis of the unicycle, see Figure 3.4. The result of the test will be the time of stabilization and the overshoot for each DOF, both in simulation and in practice.

5.1.4 Robustness to Model Error

The Robustness to model error test will be done by translating the center of mass slightly in the horizontal direction. This is be achieved by removing the weight in the battery box, both in the simulated non-linear model and the actual hardware, resulting in a translation of $\delta_{\text{COM}} \approx 13\ [mm]$. A new mathematical model is derived in the same way as described in Section 3.3, with the position of the body in the wheel frame now given by

$$\boldsymbol{p_b^w} = R_y^T \begin{bmatrix} 0 \\ \delta_{\text{COM}} \\ L_{wb} \end{bmatrix} \tag{5.6}$$

and a slight adjustment of m_b and I_b. The Maximum initial angle test will be repeated using the adjusted model.

5.1.5 Robustness to External Impulses

The Robustness to external impulses test will show how well the two methods can handle external disturbances in form of an impulse.

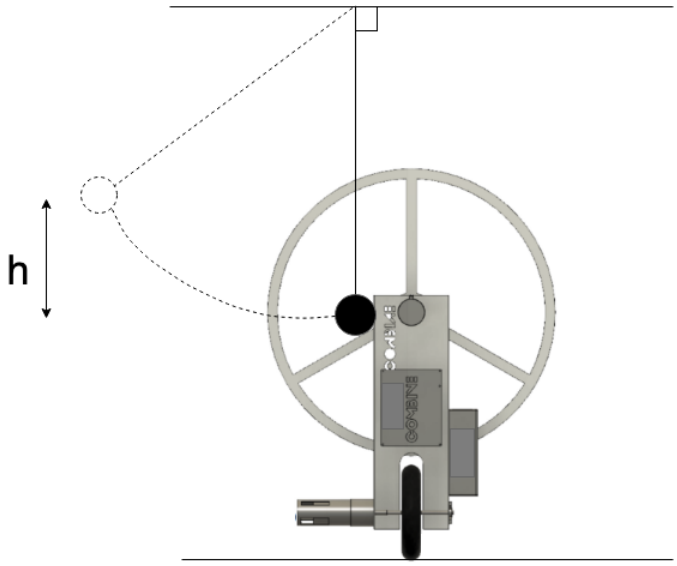


Figure 5.1: An illustration of the Impulse test setup.

In practice, the force will be applied by a ball hanging in a string from the ceiling. Before the test is initiated, the ball is moved to a fixed height along a circle with the radius equal the length of the string. Without creating a slack in the string, the ball is dropped from the initial position. It will swing down and hit the unicycle once it reaches the downward hanging position, see Figure 5.1. Since impulse can be regarded as a change in momentum, the impulse can be derived using the momentum the ball has at the time instance it hits the unicycle,

$$J_m = \int_{t_1}^{t_2} F dt \approx \Delta p = m_b v_2 - m_b v_1 \tag{5.7}$$

where J_m is the impulse, p is the momentum and m_b is the mass of the ball [13]. Assuming a perfectly inelastic collision at t_2, v_1 is the velocity of the ball at $t_1 = 0$ just before impact and $v_2 = 0$ is the velocity after the collision, at time $t_2 = \Delta t$.

In order to calculate the velocity at impact, the law of Conservation of Energy is used. Assuming the ball has no velocity before it is dropped and the height of impact is defined as zero, the law of Conservation yields

$$m_b g h = \frac{1}{2} m_b v_1^2, \tag{5.8}$$

where g is gravity and h is the height of the ball [13].

In order to keep the initial position of the unicycle as close to zero as possible, the control algorithm will be activated once the collision has occured. This is implemented in a similar way as the Maximum initial angle test, using a threshold angular velocity of the unicycle set to $\omega_{th} = 3\ [\frac{deg}{s}]$. In practice, this test will only be performed for an impulse in the Y_0 direction (reperesented as F_1 in Figure 5.2) due to the physical characteristic of the system and the fact that the hardware is sensitive to hard hits.

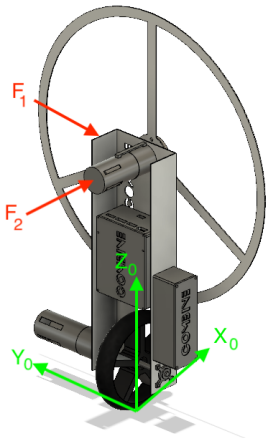


Figure 5.2: A diagram of the system showing its base frame, $X_0Y_0Z_0$ and where the two forces, F_1 and F_2 are acting on it in simulation.

In simulation, this test will be done by extending the mathematical model derived in Section 3.3 with two additional inputs representing the external forces. The forces are applied at

$$p_2 = p_d \quad \text{and} \quad p_1 = p_d + \begin{bmatrix} 0 \\ \frac{w_b}{2} \\ 0 \end{bmatrix}, \tag{5.9}$$

where $w_b = 110\ [mm]$ is the width of the body. Points p_d and p_b are the same as presented in Chapter 3, center of reaction disk and center of body respectively. Assuming the applied force is linear, by simulating a force F_e for a time Δt, the impulse can be derived using

$$J_{sim} = F_e \Delta t. \tag{5.10}$$

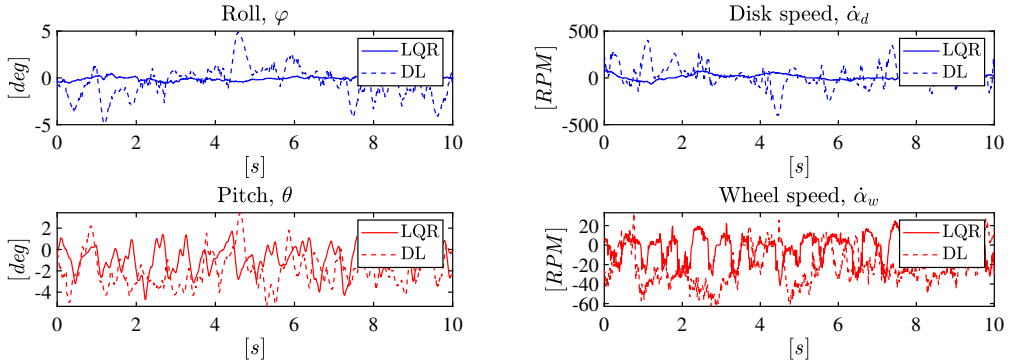
5.2 Steady State Performance

Both control methods manage to remain upright during all the executed tests. The steady state performance metrics of every test is summarized in Table 5.1. First, one can note that both the traditional LQR controller and the PPO algorithm have a fairly small mean angle in the roll direction. However, the PPO algorithm got a considerably larger standard deviation of the roll angle compared to traditional LQR

Table 5.1: Performance metrics from the Steady state performance test.

Method	Test #	Std, φ [deg]	Mean, φ [deg]	Std, θ [deg]	Mean, θ [deg]	Std, $\dot{\alpha}_d$ [RPM]	Mean, $\dot{\alpha}_d$ [RPM]	Std, $\dot{\alpha}_w$ [RPM]	Mean, $\dot{\alpha}_w$ [RPM]
LQR	1	0.1898	-0.0898	1.2058	-0.9057	26.4456	8.2613	13.2353	-3.3833
LQR	2	0.2348	0.2131	1.1274	-0.6958	26.7204	-32.9591	11.8973	-0.5704
LQR	3	0.1454	-0.0752	1.1800	-0.8256	20.6729	5.1093	12.7707	-1.1403
LQR	4	0.1854	0.2516	1.5674	-0.9115	16.9824	-35.2621	15.0449	-3.0854
LQR	5	0.1840	0.0273	1.7148	-0.8394	22.3484	-10.1699	16.3178	-1.3653
PPO	1	1.6434	-0.4834	1.5676	-1.9481	136.4362	42.0260	18.7758	-22.3214
PPO	2	1.6793	1.1328	1.5296	1.1285	123.1010	-76.0093	21.3889	15.3171
PPO	3	1.4247	0.7629	1.5172	-1.1049	116.5147	-34.7585	19.1792	-10.6706
PPO	4	1.9951	-1.2780	1.2737	-1.1804	133.3778	72.8266	15.4322	-12.8182
PPO	5	1.3123	-1.7157	1.5408	-1.5720	107.8820	111.5666	18.4815	-20.2148
LQR	**Avg**	**0.1879**	**0.0654**	**1.4831**	**-0.8356**	**22.6340**	**-13.0041**	**13.8532**	**-1.9089**
DL	**Avg**	**1.6110**	**-0.3163**	**1.4858**	**-0.9242**	**123.4623**	**23.1303**	**18.6515**	**-10.1416**

controller. This can also be seen in Figure 5.3a, in which the angle states of the unicycle are plotted separately for both methods in the Steady State Performance test. This stuttering performance of the roll angle of the PPO alogithm is caused by the behaviour of the disk. In Figure 5.3b, where the motor velocities are plotted separately for both methods during Test 1, one can see that the disk velocity of the PPO algorithm reaches a maximum velocity of $\dot{\alpha}_d \approx 400$ [rpm]. This can also be



(a) Angles of the unicycle

(b) Angular velocities of the motors

Figure 5.3: A typical result from the Steady state performance test. Angles of the unicycle and motor velocities are plotted for the traditional LQR controller in solid lines and for the PPO algorithm in dashed lines.

observed in the average result as the standard deviation of the disk velocity in the PPO algorithm is $\sigma_w \approx 123$ [RPM] compared to the traditional LQR controller which has a standard deviation of the disk velocity of $\sigma_d \approx 23$ [RPM].

Using a similar argumentation observing Table 5.1 and Figure 5.3a, one can note that the performance of the stabilization of the pitch angle is similar for the two methods. However, the PPO method stabilizes using a small constant velocity on the ground wheel. This will ultimately end up in the unicycle slowly travelling in the positive x- direction.

Utilizing the metrics from Table 5.1, the stabilization interval defined in Equation 5.5 is determined using

$$\delta_a = 2 \qquad \delta_\alpha = 130, \tag{5.11}$$

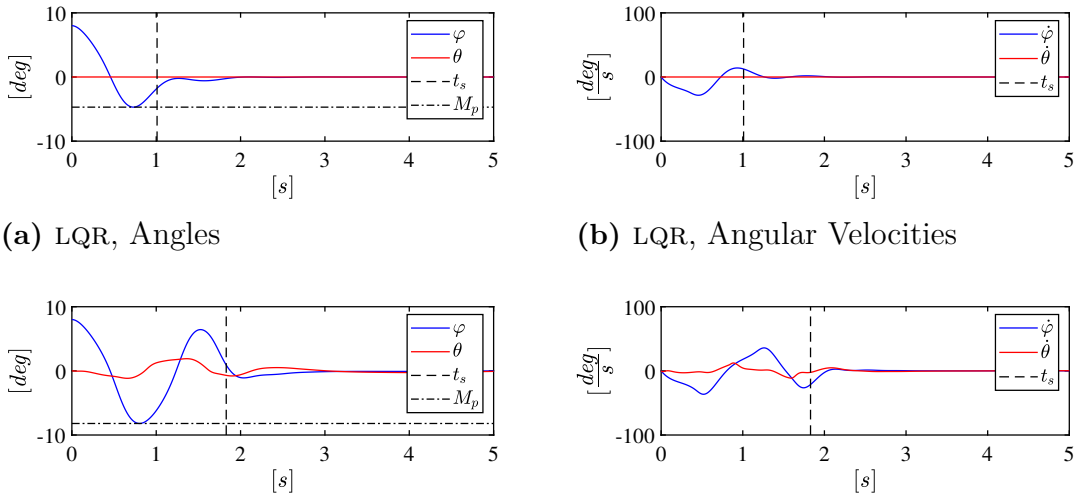
if the tested initial angle is a roll angle and

$$\delta_a = 2 \qquad \delta_\alpha = 20, \tag{5.12}$$

if the tested initial angle is a pitch angle.

5.3 Maximum Initial Angle of Stabilization

The maximum initial roll angle that both the PPO algorithm and the traditional LQR controller manage to stabilize in simulation is $\varphi_0 = 8\ [deg]$. The plots of angles and angular velocities of the unicycle and the time of stabilization t_s as well as the maximal overshoot M_p for the simulation using traditional LQR are shown in Figure 5.4a and 5.4b. The corresponding plots using PPO are shown in Figure 5.4c and 5.4d.

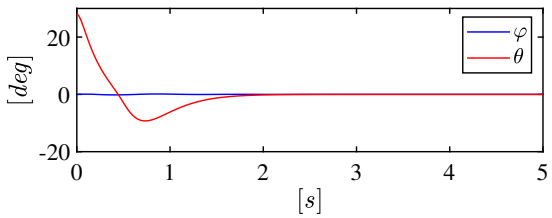


(a) LQR, Angles **(b)** LQR, Angular Velocities

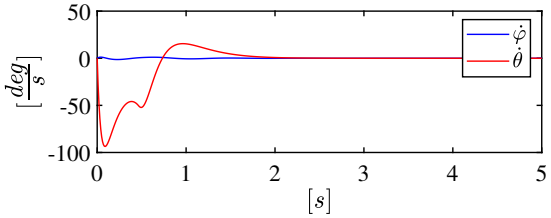
(c) PPO, Angles **(d)** PPO, Angular Velocities

Figure 5.4: Simulation results for the maximum roll angles $\varphi_0 = 8\ [deg]$ for both LQR and PPO. The states of the unicycle are shown in solid lines and the performance metrics are shown in dashed lines.

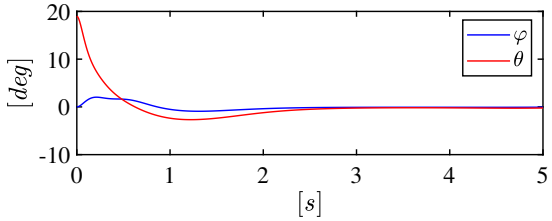
In the pitch the PPO algorithm manages to stabilize an initial angle of $\theta_0 = 19\ [deg]$ while the traditional LQR manages to stabilize an initial angle of $\theta_0 = 28\ [deg]$. In Figure 5.5a and 5.5b, the angles and the angular velocities of the traditional LQR in the simulation are shown. The corresponding plots using PPO are shown in Figure 5.5c and 5.5d.



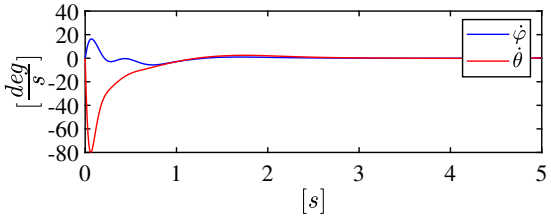
(a) LQR, Angles



(b) LQR, Angular Velocities



(c) PPO, Angles



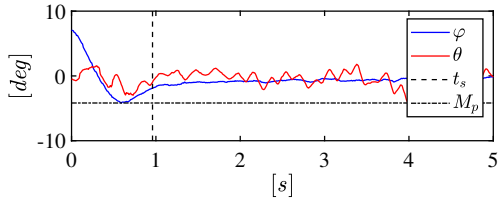
(d) PPO, Angular Velocities

Figure 5.5: Simulation results for the maximum pitch angles $\theta_0 = 19$ $[deg]$ for PPO and $\theta_0 = 28$ $[deg]$ for LQR.

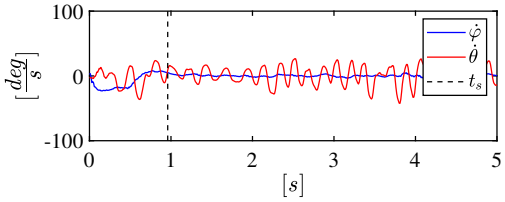
A similar result is achieved in practice. In Table C.1 in Appendix C.1, a summary of the approved tests executed for an initial roll angle of $\hat{\varphi}_0 = 7$ $[deg]$ using the traditional LQR controller can be found. The unicycle manages to stabilize 9 out of 10 attempts and it is the maximum integer roll angle that passes the test procedure requirements, see Section 5.1.2. In Table C.2, in Appendix C.1, a corresponding summary for the same initial angle using the PPO algorithm is found. The unicycle manages to stabilize 8 out of 10 attempts and turns out to be the largest initial roll angle that passes the test procedure requirements. Hence, both methods have the same maximum angle of stabilization in practice as well. In Figure 5.6, the angles and the angular velocities as well as the time of stabilization t_s and the maximal overshoot M_p from Test 4 using the traditional LQR controller and Test 9 using the PPO algorithm are shown. These tests represent the general performance of the respective control method.

In Table C.3 in Appendix C.1, a summary of the approved tests executed for an initial pitch angle of $\hat{\theta}_0 = 28$ $[deg]$ using the traditional LQR controller can be found. The method manages to stabilize 7 of 10 attempts of this angle which is the maximum initial pitch angle that passes the test procedure requirements. In Figure 5.7c and 5.7d, the angles and angular velocities for test 7 are shown which represent the general outcome.

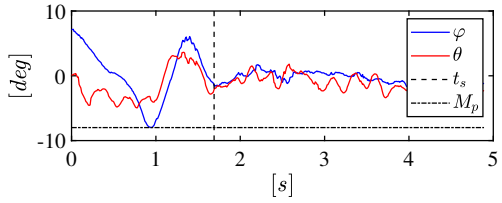
The maximum initial pitch angle that passes the test procedure requirements using the PPO algorithm is $\hat{\theta}_0 = 20$ $[deg]$ and the summary of all approved tests is shown in Table C.4 in Appendix C.1. The PPO algorithm successfully passes 7 out of 10 tests from this initial angle. In Figure 5.7a and 5.7b, the angles and angular velocities for Test 10 are shown which represent the general outcome.



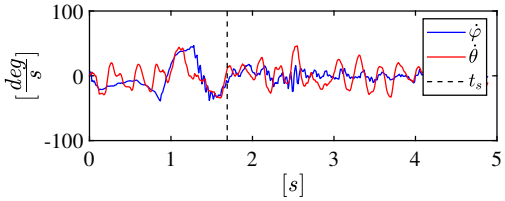
(a) LQR, Angles



(b) LQR, Angular Velocities

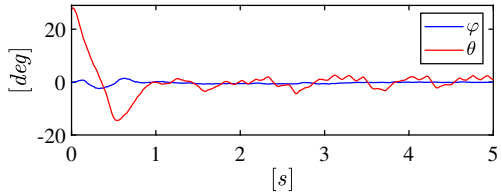


(c) PPO, Angles

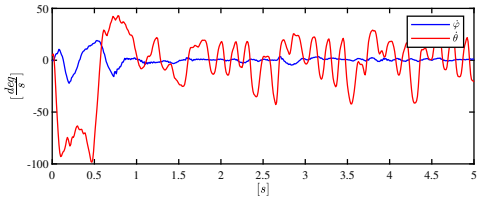


(d) PPO, Angular Velocities

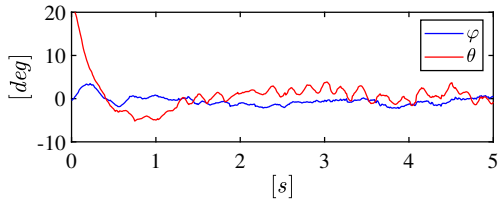
Figure 5.6: A typical result for the maximum roll angles $\varphi_0 = 7\ [deg]$ for both LQR and PPO. The states of the unicycle are shown in solid lines and the performance metrics are shown in dashed lines.



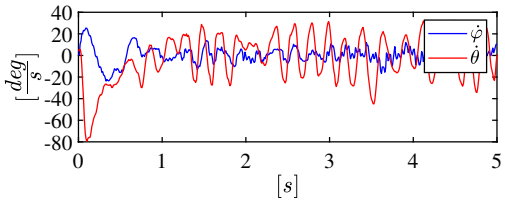
(a) LQR, Angles



(b) LQR, Angular velocities



(c) PPO, Angles



(d) PPO, Angular velocities

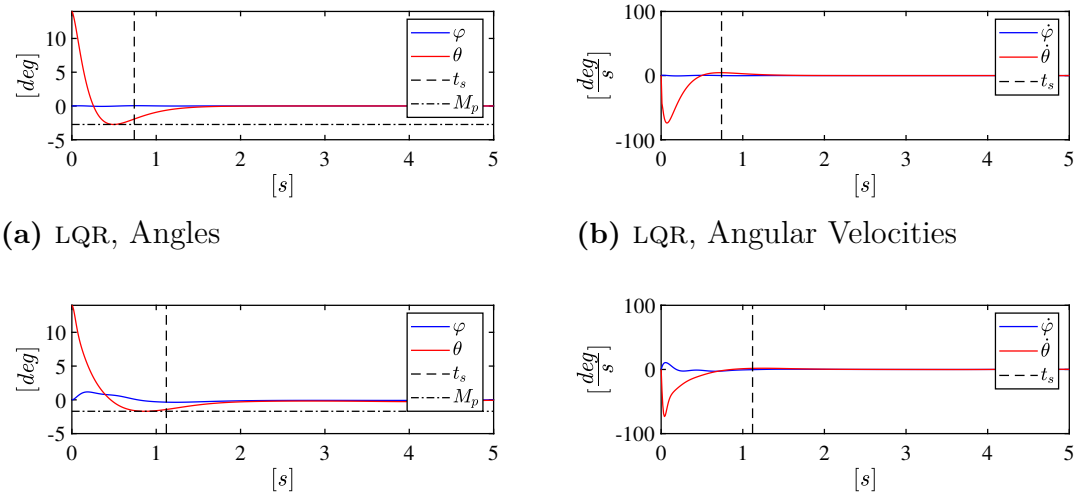
Figure 5.7: A typical result for the maximum pitch angles $\theta_0 = 28\ [deg]$ for LQR and $\theta_0 = 20\ [deg]$ for PPO.

5.4 General Performance Metrics

The fact that both control methods have an equal maximum angle of stabilization in the roll angle enables the general performance metrics of the roll to be evaluated at this angle. This is preferable due to the fairly large stabilization interval of the roll angle as defined in Equation 5.11. Hence, in simulation, the general performance metrics are evaluated at $\varphi_0 = 8\ [deg]$, see Figure 5.4. The traditional LQR controller has both a smaller time of stabilization $t_s = 1.01\ [s]$ and overshoot $M_p = 4.7\ [deg]$

compared to the PPO algorithm which has $t_s = 1.83$ $[s]$ and $M_p = 8.2$ $[deg]$.

As the two methods have different maximum pitch angles, the performance metrics in the general case will be done from an initial angle that is inside both methods stabilization interval. The chosen initial angle that will be tested is set to $\hat{\theta}_0 = 14$ $[deg]$. Using the limits defined in Equation 5.12, the results of the simulation are shown in Figure 5.8. Unlike the roll performance, the PPO algorithm has a lower overshoot of $M_p = 1.7$ $[deg]$ compared to the traditional LQR controller which has an overshoot of $M_p = 2.74$ [deg]. However, the traditional LQR controller has a smaller time of stabilization $t_s = 0.74$ $[s]$ compared to the PPO algorithm which has $t_s = 1.12$ $[s]$.



(a) LQR, Angles

(b) LQR, Angular Velocities

(c) PPO, Angles

(d) PPO, Angular Velocities

Figure 5.8: Simulation result from the General performance metric test from an initial pitch angle of $\theta_0 = 14$ [deg] for both LQR amd PPO. The states of the unicycle are shown in solid lines and the performance metrics are shown in dashed lines.

Finally, in order to be able to compare simulation and practical performance, a last simulation from an initial angle of $\varphi_0 = 7$ $[deg]$ is made. The results of the performance metrics from all simulations are summarized in Table 5.2.

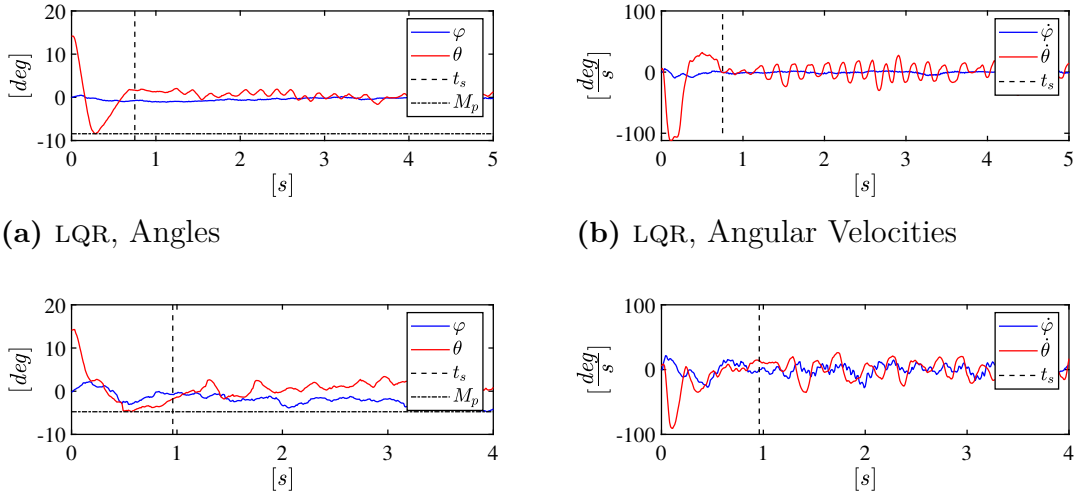
Using the practical tests from the maximum roll angle, see Figure 5.6, the performance metrics from all tests are summarized in Table C.5 in Appendix C.2. Similar to the simulations, the traditional LQR controller has both a smaller average time of stabilization $t_s \approx 0.9$ $[s]$ and overshoot $M_p \approx 4.0$ $[deg]$ compared to the PPO algorithm which has $t_s \approx 1.4$ $[s]$ and $M_p = 7.8$ $[deg]$.

The performance metrics of 10 approved tests from an initial pitch angle of $\hat{\theta}_0 = 14$ $[deg]$ are summarized in Table C.6 in Appendix C.2. A general plot of the angles and angular velocities and the performance metrics for these tests are shown in Figure 5.9. A result similar to the simulation is achieved in the pitch direction as well, as the PPO algorithm has a smaller overshoot but a larger time of stabilization

Table 5.2: General performance metrics from simulations of different initial positions.

Initial angle, $[deg]$	Method	Time of Stabilization, t_s $[s]$	Overshoot, M_p $[deg]$
$\varphi_0 = 8$	LQR	1.01	4.70
$\varphi_0 = 8$	PPO	1.83	8.20
$\theta_0 = 14$	LQR	0.74	2.74
$\theta_0 = 14$	PPO	1.12	1.70
$\varphi_0 = 7$	LQR	0.79	3.53
$\varphi_0 = 7$	PPO	0.82	4.57

in comparison to the traditional LQR controller. A table summarizing the average performance metrics of the practical tests is shown in Table 5.3.



(a) LQR, Angles

(b) LQR, Angular Velocities

(c) PPO, Angles

(d) PPO, Angular Velocities

Figure 5.9: A typical result from the General performance metrics test from an initial pitch angle of $\hat{\theta}_0 = 14$ [deg] for both LQR amd PPO. The states of the unicycle are shown in solid lines and the performance metrics are shown in dashed lines.

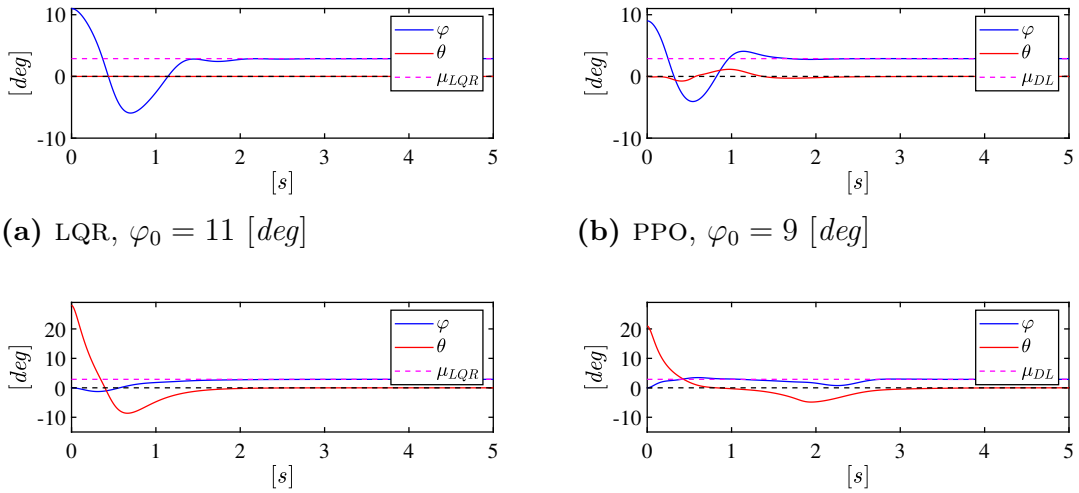
5.5 Robustness to Model Errors

Simulations of maximum angles using the mathematical model with translated center of mass for both controllers are shown in Figure 5.10. The maximum angle in a positive roll direction is to $\varphi_0 = 11$ $[deg]$ for the traditional LQR controller and $\varphi_0 = 9$ $[deg]$ for the PPO algorithm. Hence, when simulating the model with translated center of mass, the traditional LQR controller can handle a larger initial positive roll angle compared to the PPO algorithm and both methods performs better compared to the ideal simulation. However, in the negative roll direction, both controllers can only manage to stabilize from a maximum negative angle of

Table 5.3: Average general performance metrics from practical tests in different initial positions.

Initial angle, $[deg]$	Method	Time of Stabilization, t_s $[s]$	Overshoot, M_p $[deg]$
$\hat{\varphi}_0 = 7$	LQR	0.8622	4.0083
$\hat{\varphi}_0 = 7$	PPO	1.3775	7.8011
$\hat{\theta}_0 = 14$	LQR	0.7920	7.8367
$\hat{\theta}_0 = 14$	PPO	0.8330	4.4357

$\varphi_0 = -6$ $[deg]$. Furthermore, the maximum pitch angles of stabilization in simulation using the traditional LQR controller are $\theta_0 = 28$ $[deg]$ while the PPO algorithm can manage from $\theta_0 = 19$ $[deg]$. Additionally, in Figure 5.10 one can notice that both controllers stabilize around a steady state angle of $\mu_{\text{LQR}} = \mu_{\text{PPO}} = 2.88$ $[deg]$ in the roll while the pitch angle stabilizes at zero.



(a) LQR, $\varphi_0 = 11$ $[deg]$

(b) PPO, $\varphi_0 = 9$ $[deg]$

(c) LQR, $\theta_0 = 28$ $[deg]$

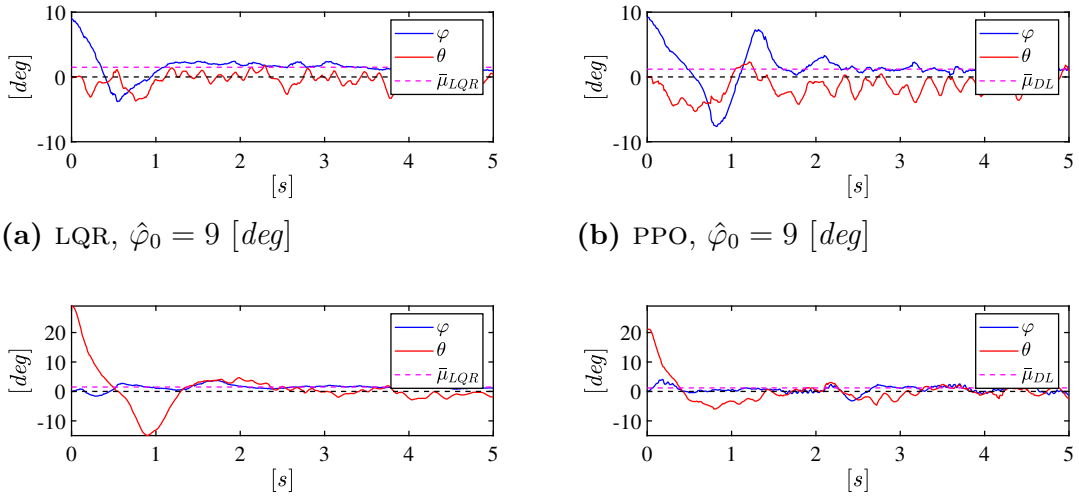
(d) PPO, $\theta_0 = 19$ $[deg]$

Figure 5.10: Simulation from the Robustness to model errors test. The angles are shown in solid lines and the dashed lines are the values which the angles are converging to.

In tables C.7- C.10 in Appendix C, results can be found from all practical tests made on the system containing the model error. The traditional LQR controller manages to stabilize in 9 out of 10 tries from an initial roll angle of $\hat{\varphi}_0 = 9$ $[deg]$ and 8 out of 10 tries from an initial pitch angle of $\hat{\theta}_0 = 29$ $[deg]$. The PPO algorithm manages to stabilize in 7 out of 10 times from an initial roll angle of $\hat{\varphi}_0 = 9$ $[deg]$ and 5 out of 10 times from an initial pitch angle of $\hat{\theta}_0 = 21$ $[deg]$.

Figure 5.11 shows a typical result of pitch and roll angles for both controllers from all performed practical tests. One can note that, as in simulation, the roll angle converges to a steady state in both methods. In order to derive the average value in

which the method seems to converge towards, one needs to define a time instance in which the controller have stabilized. Utilizing the result from Section 5.4, it can be seen that both controllers have stabilized well within the interval $t \in \begin{bmatrix} 0 & 2 \end{bmatrix}$ $[s]$. By taking the average of the roll data in the tests performed from an initial roll angle after $t = 2$ $[s]$ for both methods, one achieves the results shown in Table 5.4. The practical offsets $\bar{\mu}_{\text{LQR}} \approx 1.5$ $[deg]$ and $\bar{\mu}_{\text{PPO}} \approx 1.2$ $[deg]$ are derived as the mean offset of all tests and is plotted Figure 5.11.



(a) LQR, $\hat{\varphi}_0 = 9$ $[deg]$

(b) PPO, $\hat{\varphi}_0 = 9$ $[deg]$

(c) LQR, $\hat{\theta}_0 = 29$ $[deg]$

(d) PPO, $\hat{\theta}_0 = 21$ $[deg]$

Figure 5.11: A typical result from the Robustness to model errors test in practice. The angles are shown in solid lines and the dashed lines are the values which the angles are converging to.

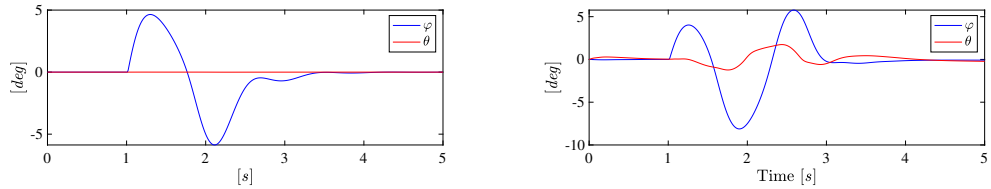
Table 5.4: The average angle of convergence from the practical Robustness to model error tests performed from an initial roll angle. The average convergence is derived to be $\bar{\mu}_{\text{LQR}} \approx 1.2$ for the traditional LQR controller and $\bar{\mu}_{\text{PPO}} \approx 1.5$ for the PPO algorithm.

Test #	1	2	3	4	5	6	7	8	9	10	**Avg**
μ_{PPO}	1.2	1.2806	1.1943	1.1814	1.9879	-0.367	-	-	1.9286	-	**1.2008**
μ_{LQR}	2.2935	0.86413	1.3702	-	1.5176	1.0713	2.472	1.5314	1.251	1.1035	**1.4972**

5.6 Robustness to External Impulses

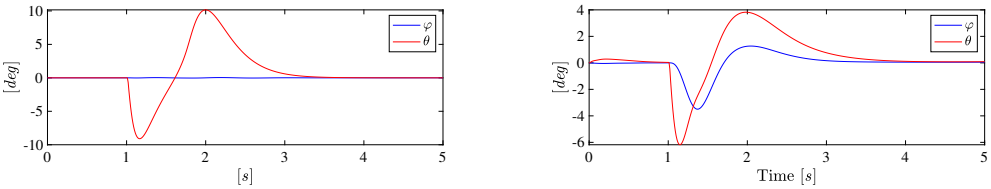
Simulation of the effects of external impulses applied along Y_0 axis as described in Figure 5.2 are presented in Figure 5.12. Similarly, Figure 5.13 shows the effect of an external impulse along the X_0 axis of both controllers. In simulation, the maximum impulse the traditional LQR controller is able to handle in the Y_0 direction (corresponding to the applied force F_1 in Figure 5.2) is $J_{\text{Y}} = 0.2$ $[Ns]$ and the corresponding maximum impulse for the PPO algorithm is $J_{\text{Y}} = 0.2$ $[Ns]$ as well. Similarly, the maximum impulse the traditional LQR controller manages to stabilize in the X_0 direction (corresponding to the force F_2 in Figure 5.2) is $J_{\text{X}} = 0.49$ $[Ns]$

compared to the PPO algorithm which manages to stabilize a maximum impulse of $J_x = 0.34$ $[Ns]$. Thus, the two algorithms are equally robust in the roll angle but the traditional LQR controller is more robust in the pitch angle.



(a) LQR **(b)** PPO

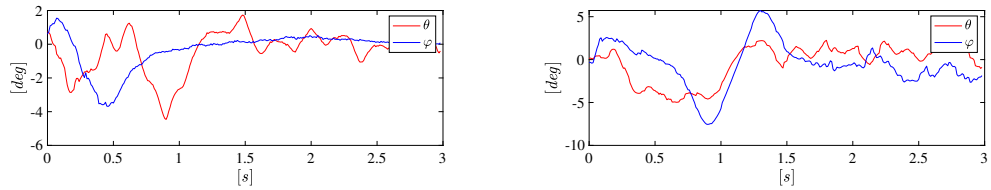
Figure 5.12: Result of the system angles from simulations of the effect of maximum external impulse parallel to the Y_0 axis.



(a) LQR **(b)** PPO

Figure 5.13: Result of the system angles from simulations of the effect of maximum external impulse parallel to the X_0 axis.

Tables C.11 and C.12 in Appendix C show a summary of initial conditions for each performed practical test as well as an index indicating whether or not the system passes the test by managing to stay upright after the impulse is applied. By increasing the height of the initial position of the ball in steps of $\Delta h = 10$ $[mm]$, the maximum impulse in the Y_0 direction is determined to $J_Y \approx 0.2$ $[Ns]$ for both methods. The traditional LQR controller manages to stabilize in 6 out of 10 tries and the PPO algorithm manages to stabilize in 7 out of 10 tries. Figure 5.14 shows a general reaction of the angular response from the maximum impulse disturbance.



(a) LQR **(b)** PPO

Figure 5.14: A typical result of the system angles from the Robustness to external imupses test from an external impulse parallel to the Y_0 axis.

Large initial states in Tables C.11 and C.12 is due to the large impulse of the ball generating a fast change in the states. When the controller starts it has therefore already gotten a high angular velocity from the hit.

6
Discussion

This chapter will provide an analysis and a general discussion on the test results presented in the previous chapter. Furthermore, an attempt of generalizing the results to cast light on a wider perspective when comparing the two methods will be done. Finally, future possibilities will be briefly addressed.

6.1 General Comparison Between the Methods

Even though both the traditional LQR controller and the PPO algorithm successfully stabilize the unicycle within an interval of stabilization, there are many interesting aspects in the performance and robustness of the methods to discuss. As described in Section 5.2, the traditional LQR controller has a better steady state performance around the X_0 axis of the unicycle. This performance advantage also appear in the General performance metrics tests, both in simulation and practice (see Tables 5.2 and 5.3). Furthermore, one can also notice a large standard deviation of the time of stabilization in the PPO algorithm during the roll tests (see Table C.5 in Appendix C.2). This is a direct implication from the large standard deviation of the roll angle in the stable state. However, when it comes to testing the limits of the roll, such as the largest initial angle, the PPO algorithm performs equally well as the traditional LQR controller. This is the case both in simulation and in practice, see section 5.3. The two algorithms also perform equally well around the X_0 axis when it comes to the largest external disturbances that the two methods can handle, which can be seen in Section 5.6.

The similar behaviour in the systems extreme conditions in the roll indicates a hardware limit of the unicycle. Once exposed to these conditions, the disk motor velocity saturates and no more torque can be applied. The results from the Robustness to model error test in Section 5.5 provide further support for the hardware limit argument of the unicycle. By removing the stabilization weight of the unicycle, both control methods manage to stabilize a larger maximum roll angle. However, both methods have a smaller maximum roll angle in the negative direction when tested in simulation. This is expected due to the small positive torque that is introduced by gravity due to the translated center of mass. Similarly, an improved maximum pitch angle is expected for both methods due to the decreased total weight of the system. In simulation this is only the case regarding the traditional LQR controller. In fact, the maximal initial angle of stabilization around the Y_0 axis using the PPO algorithm

decreases compared to the ideal model. Hence, it can be argued that the limits of stabilization is hardware related for both methods in the roll angle while the PPO algorithm is limited in the pitch angle due to its stabilization method. Therefore, in order to achieve a larger angle of stabilization and to reject a larger external disturbance around the X_0 axis of the unicycle, the hardware section should be adapted for the system. For further reading on suggested adjustments, see Section 6.4.

One can also note that while the traditional LQR controller manages to stabilize a roll angle up to $\varphi_0 = 11$ $[deg]$ when simulating the model with an adjusted center of mass, the PPO only manages an angle up to $\varphi_0 = 9$ $[deg]$. This is most probably due to that the training of the PPO algorithm was made with an initial maximum angle of $\varphi_0 = 9$ $[deg]$ (see Section 4.2.3). Hence, the argument can be made that in order for the PPO algorithm to perform, it relies on having experienced a similar scenario during training.

Another interesting thing to note from the stabilization of the maximum initial roll angle using the PPO algorithm, is that the overshoot is larger than the initial angle itself. This can be seen in Figure 5.6c, where the roll angle exceeds $\varphi \approx 8.5$ $[deg]$. The reason why the method manages to stabilize an overshoot larger than the maximum initial angle is due to the disk having a wider range to accelerate, as the velocity of the disk at the time instance of the overshoot is larger than zero in the opposite direction. This behaviour is undoubtedly not optimal and indicates that the PPO algorithm has converged to a local optimum solution. In this local optima, one should as well note that the pitch and roll seem to be more coupled than the traditional LQR controller anticipates. This can be seen as the PPO algorithm relies more on movement in roll in order to stabilize its pitch and wise versa. An example of this can be observed in most figures, e.g. Figure 5.4c.

While the performance differences between the two methods is mainly shown in small roll angles, they have a similar performance in small pitch angles, as can be seen in Section 5.2. When comparing the general performance metrics about the Y_0 axis of the two methods, the traditional LQR controller has a faster time of stabilization than the PPO algorithm. However, the overshoot of the PPO algorithm is smaller than that of the LQR controller, which is the case in both simulation and practice as can be seen in Table 5.2 and 5.3, respectively. Hence, the control around the Y_0 axis using the PPO algorithm is more conservative than the traditional LQR controller. This is presumably the reason the PPO algorithm have a worse performance in the extreme conditions of the pitch angle. It manages to stabilize both a smaller maximum pitch angle and a smaller external disturbance in comparison to the traditional LQR controller. Consequently, one can make the argument that the robustness detriment of the PPO algorithm is due to the conservatism of the local optima solution.

This conservatism of the PPO method puts an expectation that it should have a better steady state performance compared to the more aggressive traditional LQR controller. However, this is not the case, as can be seen in Table 5.1. In fact, the

traditional LQR controller seems to perform slightly better as it has both a smaller standard deviation and mean value of the wheel motor. This indicates a minor inconsistency of the PPO algorithm.

Finally, even though the two methods have similar extreme limits around the X_0 axis, the traditional LQR controller successfully stabilizes a larger amount of the tests in these scenarios, which can be seen in Appendix C. Altogether, the traditional LQR controller seems to outperform the PPO algorithm in most aspects.

6.2 Comparison Between Simulation and Practice

To begin with, a similar behaviour in simulation as well as the real physical system can be observed for both methods in most tests. The shape of curves and the order of magnitude seen in plots indicate that the dynamics of the mathematical model matches the real system, which further support the conclusion from Section 3.5. A clear example of this can be seen by comparing the angular velocities in Figure 5.5 to Figure 5.7. In both figures, the angular velocity of the traditional LQR controller takes a small dive on its way up from the initial angle to a stable performance in both simulation and practice. Furthermore, the rise in the angular velocity around X_0 axis in the PPO can as well easily be seen in both simulation and the real system. However, one has to keep in mind that there are several indications that the model can be improved as well. Even though the shape of the dynamics seem to match relatively well there are mismatching magnitude of some of the performance metrics when comparing the simulation results to the real system. For instance, when comparing the general performance metrics in Table 5.2 and 5.3, the amplitude of the overshoot around the Y_0 axis is more than double in practice compared to in simulation for both methods. The same applies for the PPO algorithm in the X_0 direction. Furthermore, the results of the Maximum initial angle test and the Robustness to model error test do not yield identical results in simulation and in practice. In general, both controllers stabilize 1 $[deg]$ more around the X_0 axis and 1 $[deg]$ less around the Y_0 axis in practice than in simulation. This may indicate that the parameters for the disk motor is "oversized" and the parameters in the wheel motor is "undersized". However, the magnitude of the external impulse test matched perfectly in simulation and in practice. Hence, to improve the model, parameter identification should be made which is left for future work.

More importantly, even though the magnitude of all performance metrics did not match perfectly, the proportion of performance when comparing the two methods matched fairly well. For instance, both methods stabilized equal maximum angles around the X_0 axis and the more conservative behaviour of the PPO algorithm in comparison to the traditional LQR controller around the Y_0 axis is evident both in practice and simulation. This indicates that both methods managed the transfer of the controller from simulation to the real physical system fairly well.

As described in Section 4.2.3, the main change to successfully transfer the PPO algorithm to the real physical system was to train it with noise on the state observations. By observing the results of the Maximum initial angle test and the Robustness to model error test, one could argument that the PPO algorithm is a bit more prepared for the transfer to practice. Around the Y_0 axis, the PPO algorithm manages 1 respectively 2 [*deg*] more in practice than simulation compared to the LQR for which the corresponding values were 0 and 1 [*deg*] more. Even though this difference could be due to other factors, it is still interesting to note that the PPO algorithm can more easily be prepared for the real system by adding nonlinearites as external disturbances and nonlinear filters to the training. This is left for future work and is further discussed in Section 6.4.

6.3 Generalization of the Results

While evaluating the achieved results in Chapter 5, it is important to keep in mind that the two derived controllers are not unique. By tuning the weights and parameters in the traditional LQR controller as well as the PPO algorithm, different observations and results could be achieved. For instance, by further tuning, the conservatism of the PPO algorithm could most likely be adjusted. However, in this project, a significant amount of time was invested particularly on the PPO algorithm compared to the traditional LQR controller. The effect of a change on a specific parameter may be hard to predict and the training to find out is time consuming.

It is as well important to mention that this comparison is only an example on one particular system. The inverted pendulum is a standard subject of control theory and has a well known optimal behaviour. As mentioned earlier, the PPO algorithm shows a different and somewhat unexpected behaviour around the X_0 axis of the unicycle, which indicates the the PPO algorithm converged to a local optima. It can therefore be suggested that for systems with no clear or even unknown optimal solution or for systems that one wishes to explore, the PPO algorithm could be a feasible solution.

On the other hand, as mentioned in section 6.1, the PPO algorithm relies on having experienced a scenario during training similar to that it is currently encountering. For systems with large state spaces or unknown limits of the states, a DL based control might not be desirable. If one cannot define the limits of the state space in which the system will be deployed in, one can not guarantee any performance.

6.4 Future Work

Due to the dynamics of the unicycle having an unstable stabilization point, the system is an interesting subject for comparing control methods and hence the possibilities of future work on the system are many. To begin with, one should adjust the hardware section in order to improve the performance in the extreme conditions

in the roll angle. One way to achieve this would be to increase the inertia of the disk to make sure that the motor speed does not saturate. However, one should keep in mind that increasing the inertia of the disk will increase the total mass of the system, which will most likely result in a worse performance about the Y_0 axis.

There are many ways to improve the performance of the PPO algorithm on the unicycle. By extending its reward function to include a term representing the energy loss, one could imagine it to behave more smoothly and reduce large jumps in input and thus a high acceleration of the disk. Furthermore, by introducing the algorithm to all filters and disturbances in the training environment, the PPO algorithm should become more robust. The ideal way of doing this would be to fine tune it on the actual hardware. That way it could get to know the systems characteristics on more detail and better adjust its policy towards a more optimal control. A performance comparison between a controller trained in simulation and a controller fine tuned on hardware would have been of great interest.

Finally, in order to increase the robustness of the unicycle using traditional control theory, a robust controller such as the $\mathcal{H}_\infty$ controller could be implemented. This should end up in a better performance to disturbances and noise.

7
Conclusion

The constructed unicycle managed to stabilize within an interval of stabilization and remained stable using both methods. Even though the mathematical model resembled the unicycle fairly well, it was not perfect and there is room for improvements. Despite this, the PPO policy trained in simulation using noise on the observations transferred to the real system successfully. Transferring policies from simulation to a real physical system is considered a viable option for systems in which training from scratch on the physical system itself is practically infeasible. An interesting aspect would be to take a PPO algorithm trained in simulation and fine tune it on hardware subject to external disturbances, which is left for future research.

The traditional LQR controller proved to outperform the PPO algorithm in most perspectives where the hardware of the unicycle did not set any limitations. This was the case both in simulation and practice. Similarly, the traditional LQR controller also proved to be more robust to external disturbances where the hardware limits were not an issue. This is likely due to the local optima the PPO algorithm converged to in which it displayed a somewhat unexpected way to stabilize the roll angle of the unicycle. This innovative behaviour might be desirable for some systems, yet for systems with a known global optima, the traditional control methods are recommended, if applicable.

Finally, the robustness advantage of the traditional LQR controller was most probably due to the conservatism of the achieved PPO solution. As both methods handled the enforced model errors similarly, and both methods seemed to have a similar performance in simulation as in practice there were no signs of uncertain behaviours regarding the robustness of the PPO algorithm compared to the traditional LQR controller. In fact, by introducing the PPO algorithm to all filters and disturbances, the difference between simulation and practice might be reduced for the method, which is left as a future investigation.

Bibliography

[1] R. Matthew Kretchmar. "A Synthesis of Reinforcement Learning and Robust Control Theory". PhD thesis. Colorado State University, 2000. URL: http://www.cs.colostate.edu/~anderson/res/rl/matt-diss.pdf.

[2] Dario Amodei et al. "Concrete Problems in AI Safety". In: *CoRR* abs/1606.06565 (2016). arXiv: 1606.06565. URL: http://arxiv.org/abs/1606.06565.

[3] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning." In: *CoRR* abs/1509.02971 (2015). arXiv: 1509.02971 [cs.LG]. URL: https://arxiv.org/abs/1509.02971.

[4] Mnih Volodymyr et al. "Human-level control through deep reinforcement learning." In: *Nature* 7540 (2015), p. 529. ISSN: 0028-0836.

[5] J. Fu, S. Levine, and P. Abbeel. "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016, pp. 4019–4026. DOI: 10.1109/IROS.2016.7759592.

[6] Paul F. Christiano et al. "Transfer from Simulation to Real World through Learning Deep Inverse Dynamics Model". In: *CoRR* abs/1610.03518 (2016). arXiv: 1610.03518. URL: http://arxiv.org/abs/1610.03518.

[7] A. Nagabandi et al. "Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning". In: *arXiv e-prints* (Mar. 2018). arXiv: 1803.11347.

[8] J. Hwangbo et al. "Control of a Quadrotor With Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 2.4 (Oct. 2017), pp. 2096–2103. DOI: 10.1109/LRA.2017.2720851.

[9] Jesper Hove Jørgensen and Jonas Ørndrup Nielsen. "Non-linear Control and Machine Learning on an Inverted Pendulum on a Cart". MA thesis. Fredrik Bajers vej 7 9220 Aalborg: Aalborg University, June 2018. URL: https://projekter.aau.dk/projekter/files/281331409/Non_linear_Control_and_Machine_Learning_on_an_Inverted_Pendulum_on_a_Cart.pdf.

[10] Indrazno Siradjuddin et al. "State space control using LQR method for a cart-inverted pendulum linearised model". In: *IJMME-IJENS* 17 (Feb. 2017), pp. 119–126.

[11] J. Lee, S. Han, and J. Lee. "Decoupled Dynamic Control for Pitch and Roll Axes of the Unicycle Robot". In: *IEEE Transactions on Industrial Electronics* 60.9 (Sept. 2013), pp. 3814–3822. ISSN: 0278-0046. DOI: 10.1109/TIE.2012.2208431.

[12] Anders Boström. *Lecture notes in MMA092, Rigid body dynamics*. Chalmers University of Technology, Sweden. Oct. 2016.

[13] Reza N. Jazar. *Advanced dynamics. [electronic resource] : rigid body, multi-body, and aerospace applications.* Wiley, 2011. ISBN: 9780470398357.

[14] H. Petkov Petko, N. Slavov Tsonyo, and K. Kralev Jordan. "4.2.1 Discrete-Time LQG Controller." In: *Design of Embedded Robust Control Systems Using MATLAB /Simulink*. Institution of Engineering and Technology, 2019. ISBN: 978-1-5231-2106-9.

[15] Karl J. Åström and Richard M. Murray. *Feedback systems. [electronic resource] : an introduction for scientists and engineers.* Princeton, N.J. ; Woodstock : Princeton University Press, c2008., 2008. ISBN: 1400828732.

[16] Manon Kok, Jeroen Hol, and Thomas Schön. "Using Inertial Sensors for Position and Orientation Estimation". In: *Foundation and Trends of Signal Processing* 11 (Apr. 2017). DOI: 10.1561/2000000094.

[17] Simo Särkkä. *Bayesian Filtering and Smoothing*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2013.

[18] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Cambridge, MA : The MIT Press, 2018.

[19] Yoshua Bengio. *Introduction to Gradient-Based Learning, Notes de cours IFT6266 Hiver 2010*. Online lecture. Apr. 2010. URL: http://www.iro.umontreal.ca/~pift6266/H10/notes/gradient.html#gradient.

[20] Polk and Seifert. *Cognitive modeling*. Ed. by Thad et al. Bradford Books. Cambridge, Mass. : MIT Press, 2002.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[22] Knut Hinkelmann. *Neural Networks*. Online lecture. 2018. URL: http://didattica.cs.unicam.it/lib/exe/fetch.php?media=didattica:magistrale:kebi:ay_1718:ke-11_neural_networks.pdf.

[23] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[24] Gabriel Perreira Das Nevas. "Modelling, Construction and Control of a Self-balancing Unicycle". MA thesis. Universidade de São Paulo, 2017. URL: https://www.teses.usp.br/teses/disponiveis/3/3139/tde-07112017-082249/publico/GabrielPereiradaNevesCorr17.pdf.

[25] *LSM9DS1*. Datasheet. Rev. 3. ST Microelectronics. Mar. 2015. URL: https://www.st.com/resource/en/datasheet/DM00103319.pdf.

[26] *ESP32 - WROOM-32*. Datasheet. Rev. 2.8. Espressif Systems. Jan. 2019. URL: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf.

[27] *639282*. Datasheet. Robot Zone. URL: https://www.robotshop.com/media/files/pdf2/sct-1021.pdf.

[28] *Planetary Gear Motor - PGM36 26.9:1*. Datasheet. Lynxmotion. URL: http://www.lynxmotion.com/p-1079-planetary-gear-motor-pgm36-2691.aspx.

[29] *MD10C*. Datasheet. Cytron Technologies. URL: https://www.robotshop.com/media/files/pdf2/cyt-132-v2.3.pdf.

[30] Graham W. Griffiths. *1.3.1 RK Coefficients*. Cambridge University Press, 2016. ISBN: 978-1-107-11561-3. URL: https://app.knovel.com/hotlink/khtml/id:kt010YD1R4/numerical-analysis-using/rk-coefficients.

[31] Huibert Kwakernaak and Raphael Sivan. *Linear optimal control systems.* Wiley, 1972. ISBN: 0-471-51110-2.

[32] Andrew Ilyas et al. "Are Deep Policy Gradient Algorithms Truly Policy Gradient Algorithms?" In: *CoRR* abs/1811.02553 (2018). arXiv: 1811.02553. URL: http://arxiv.org/abs/1811.02553.

A

Hardware Parameters

The radius of the driving wheel is $r_w = 0.072$ $[m]$. The length between the wheel frame and the body frame is $L_{wb} = 0.1292$ $[m]$ and the length between the wheel frame and the disk frame is $L_w d = 0.3006$ $[m]$.

A.1 Hardware Parameters for the Wheel

The wheel (drive wheel, hub, shaft and adapter included) got the inertia matrix

$$I_w = \begin{bmatrix} 0.3882 & -0.0000 & -0.0000 \\ -0.0000 & 0.6889 & 0.0000 \\ -0.0000 & 0.0000 & 0.3882 \end{bmatrix} \cdot 1 \times 10^{-3} \ [kgm^2] \tag{A.1}$$

and the mass $m_w = 0.3047$ $[kg]$.

A.2 Hardware Parameters for the Body

The body of the unicycle (body itself, disk motor, wheel motor, bearing, battery, stabilization weight, battery box, electronics and electronics box included) got the inertia matrix

$$\begin{bmatrix} 28.9000 & 0.0252 & 1.1860 \\ 0.0252 & 21.0200 & 4.9580 \\ 1.1860 & 4.9580 & 9.1760 \end{bmatrix} \cdot 1 \times 10^{-3} \ [kgm^2] \tag{A.2}$$

and the mass $m_b = 1.8040$ $[kg]$.

A.3 Hardware Parameters for the Disk

The disk (disk and diskhub included) got the inertia matrix

$$\begin{bmatrix} 3.9480 & -0.0000 & -0.0001 \\ -0.0000 & 1.9800 & -0.0004 \\ -0.0001 & -0.0004 & 1.9690 \end{bmatrix} \cdot 1 \times 10^{-3} \ [kgm^2] \tag{A.3}$$

and the weight $m_d = 0.2045$ $[kg]$.

A.4 Hardware Parameters for the Motors

The resistance of the wheel and disk motor respectively is $R_{a,w} = 4.8$ [Ω] and $R_{a,w} = 0.6$ [Ω].

As the current draw of the motor is small during no load, the motor velocity constant can be estimated by

$$K_u = \frac{Rated\ Voltage}{No\ load\ speed}. \tag{A.4}$$

For the wheel motor, this result in $K_{u,w} = 1.0709$ [Vs] and for the disk motor it result in $K_{u,d} = 0.2622$ [Vs].

From $T_m(t) = K_m i_a(t)$, the motor torque constant can be estimated according to

$$K_m = \frac{stall\ torque}{stall\ current}. \tag{A.5}$$

For the wheel motor, this result in $K_{m,w} = 0.6119$ [$\frac{Nm}{A}$] and for the disk motor it result in $K_{m,d} = 0.1077$ [$\frac{Nm}{A}$].

B

Traditional Control Method

B.1 Continuous Time System Matrices

The continuous time LTI system matrices are given by

$$A = \begin{bmatrix} -27.6628 & -0.0002 & 0.0000 & 0.0000 & -0.0399 & 133.8083 \\ -0.0019 & -2.9326 & 0.0000 & 0.0000 & -34.1926 & 0.0153 \\ 0.0019 & 0.1652 & 0.0000 & 0.0000 & 34.1926 & -0.0152 \\ -10.1012 & -0.0001 & 0.0000 & 0.0000 & -0.0238 & 79.9978 \\ 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 \end{bmatrix} \tag{B.1}$$

and

$$B = \begin{bmatrix} 25.8302 & 0.0007 \\ 0.0018 & 11.1837 \\ -0.0018 & -0.6299 \\ 9.4320 & 0.0004 \\ 0.0000 & 0.0000 \\ 0.0000 & 0.0000 \end{bmatrix}. \tag{B.2}$$

B.2 Discrete Time System Matrices

Using $T_s = 0.01$, the discrete time equivalence of the system matrices are given by

$$A_d = \begin{bmatrix} 0.7581 & -0.0000 & -0.0000 & 0.0061 & -0.0003 & 1.1706 \\ -0.0000 & 0.9711 & -0.0017 & 0.0000 & -0.3372 & 0.0001 \\ 0.0000 & 0.0016 & 1.0017 & -0.0000 & 0.3418 & -0.0001 \\ -0.0884 & -0.0000 & -0.0000 & 1.0038 & -0.0002 & 0.7392 \\ 0.0000 & 0.0000 & 0.0100 & -0.0000 & 1.0017 & -0.0000 \\ -0.0005 & -0.0000 & -0.0000 & 0.0100 & -0.0000 & 1.0038 \end{bmatrix} \tag{B.3}$$

and

$$B_d = \begin{bmatrix} 0.2258 & 0.0000 \\ 0.0000 & 0.1102 \\ -0.0000 & -0.0062 \\ 0.0825 & 0.0000 \\ -0.0000 & -0.0000 \\ 0.0004 & 0.0000 \end{bmatrix}. \tag{B.4}$$

B.3 Extended Discrete Time System Matrices

The discrete time system matrices extended with the filter output state is given by

$$\tilde{A}_d = \begin{bmatrix} 0.7581 & -0.0000 & -0.0000 & 0.0061 & -0.0003 & 1.1706 & 0.2258 \\ -0.0000 & 0.9711 & -0.0017 & 0.0000 & -0.3372 & 0.0001 & 0.0000 \\ 0.0000 & 0.0016 & 1.0017 & -0.0000 & 0.3418 & -0.0001 & -0.0000 \\ -0.0884 & -0.0000 & -0.0000 & 1.0038 & -0.0002 & 0.7392 & 0.0825 \\ 0.0000 & 0.0000 & 0.0100 & -0.0000 & 1.0017 & -0.0000 & -0.0000 \\ -0.0005 & -0.0000 & -0.0000 & 0.0100 & -0.0000 & 1.0038 & 0.0004 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.9000 \end{bmatrix} \tag{B.5}$$

and

$$\tilde{B}_d = \begin{bmatrix} 0.0000 & 0.0000 \\ 0.0000 & 0.1102 \\ 0.0000 & -0.0062 \\ 0.0000 & 0.0000 \\ 0.0000 & -0.0000 \\ 0.0000 & 0.0000 \\ 0.1000 & 0.0000 \end{bmatrix} \tag{B.6}$$

C

Detailed Test Results

C.1 Test Results for Maximum Initial Angle Tests

Table C.1: Test results using LQR from initial angle test $\hat{\varphi}_0 = 7\ [deg]$.

Test #	Stabilized	$\varphi_0\ [deg]$	$\dot{\varphi}_0\ [\frac{deg}{s}]$	$\theta_0\ [deg]$	$\dot{\theta}_0\ [\frac{deg}{s}]$
1	Yes	7.1052	4.1869	0.0535	1.1411
2	Yes	7.1144	3.3161	0.0728	0.8875
3	Yes	7.0715	3.8403	-0.1604	-0.9629
4	Yes	7.0509	3.2009	-0.5068	-2.7123
5	Yes	7.0708	3.0783	0.0115	0.3690
6	Yes	7.1114	3.5945	0.3593	-0.0490
7	Yes	7.1984	3.1718	0.0457	1.6010
8	No	7.0967	4.2119	-0.2538	-0.2606
9	Yes	7.0504	3.3495	-0.1353	0.8463
10	Yes	7.1211	3.8435	-0.3447	-2.2228

Table C.2: Test results using PPO from initial angle test $\hat{\varphi}_0 = 7\ [deg]$.

Test #	Stabilized	$\varphi_0\ [deg]$	$\dot{\varphi}_0\ [\frac{deg}{s}]$	$\theta_0\ [deg]$	$\dot{\theta}_0\ [\frac{deg}{s}]$
1	Yes	7.0514	3.3407	-0.2657	-0.3589
2	No	7.0514	3.8598	-0.4071	-1.6635
3	Yes	7.0522	3.0107	-0.0914	0.3628
4	Yes	7.0559	3.6560	-0.1477	-0.1164
5	Yes	7.1392	3.1307	-0.1906	0.4221
6	Yes	7.1309	3.4314	-0.0470	0.7162
7	Yes	7.0844	3.9300	0.0585	-0.2538
8	Yes	7.1133	3.2387	-0.0468	-0.1047
9	Yes	7.0925	4.2972	0.1114	1.4291
10	No	7.0173	3.2860	-0.0493	0.2614

Table C.3: Test results using LQR from initial angle test $\hat{\theta}_0 = 28\ [deg]$.

Test #	Stabilized	$\theta_0\ [deg]$	$\dot{\theta}_0\ [\frac{deg}{s}]$	$\varphi_0\ [deg]$	$\dot{\varphi}_0\ [\frac{deg}{s}]$
1	Yes	28.2253	3.0676	-0.0054	1.3140
2	No	27.9673	3.1915	-0.3999	-0.6628
3	No	28.1648	4.0687	-0.0646	-1.7452
4	Yes	28.2425	4.1090	-0.5752	-4.0893
5	No	27.9154	4.3254	-0.0370	-1.0390
6	Yes	28.0882	5.2698	-0.5945	-2.6924
7	Yes	28.0506	4.2304	-0.1830	-0.4664
8	Yes	28.0979	4.2691	-0.1054	0.3521
9	Yes	27.9750	3.5655	-0.0965	-1.0534
10	Yes	28.0490	3.5680	-0.2165	0.7503

Table C.4: Test results using PPO from initial angle test $\hat{\theta}_0 = 20\ [deg]$.

Test #	Stabilized	$\theta_0\ [deg]$	$\dot{\theta}_0\ [\frac{deg}{s}]$	$\varphi_0\ [deg]$	$\dot{\varphi}_0\ [\frac{deg}{s}]$
1	Yes	19.9644	3.8947	0.3038	-2.4489
2	Yes	20.2951	4.8152	-0.0951	-1.1272
3	No	20.0236	3.5445	-0.1231	0.2723
4	Yes	20.1037	3.2248	0.1716	3.2244
5	No	20.2785	3.8873	-0.2754	-2.1412
6	Yes	20.2809	3.2932	0.2840	4.5698
7	No	20.1519	3.3231	0.6834	9.3101
8	Yes	20.0959	3.6069	-0.2447	-0.3503
9	Yes	19.9917	3.0883	-0.2293	-2.4983
10	Yes	20.2117	3.8841	-0.3375	0.3078

C.2 Test results for General Performance Metrics

Table C.5: Performance metrics from initial angle test $\hat{\varphi}_0 = 7\ [deg]$.

Method	Test #	$t_s\ [s]$	$M_p\ [deg]$	$\varphi_0\ [deg]$	$\dot{\varphi}_0\ [\frac{deg}{s}]$
LQR	1	0.81	4.0548	7.1052	4.1869
LQR	2	0.87	4.0057	7.1144	3.3161
LQR	3	0.82	4.1809	7.0715	3.8403
LQR	4	0.96	4.1780	7.0509	3.2009
LQR	5	0.85	3.5717	7.0708	3.0783
LQR	6	0.83	3.8602	7.1114	3.5945
LQR	7	0.77	3.7210	7.1984	3.1718
LQR	8	-	-	7.0967	4.2119
LQR	9	0.82	4.1199	7.0504	3.3495
LQR	10	1.03	4.3826	7.1211	3.8435
PPO	1	1.23	8.5361	7.0514	3.3407
PPO	2	-	-	7.0514	3.8598
PPO	3	0.98	7.1959	7.0522	3.0107
PPO	4	1.43	7.7097	7.0559	3.6560
PPO	5	1.69	7.6584	7.1392	3.1307
PPO	6	1.06	8.2362	7.1309	3.4314
PPO	7	1.55	7.3143	7.0844	3.9300
PPO	8	1.39	7.7643	7.1133	3.2387
PPO	9	1.69	7.9938	7.0925	4.2972
PPO	10	-	-	7.0173	3.2860
LQR	**Avg**	**0.8622**	**4.0083**	**7.0993**	**3.5091**
PPO	**Avg**	**1.3775**	**7.8011**	**7.0900**	**3.5044**

Table C.6: Performance metrics from initial angle test $\hat{\theta}_0 = 14$ $[deg]$.

Method	Test #	t_s $[s]$	M_p $[deg]$	θ_0 $[deg]$	$\dot{\theta}_0$ $[\frac{deg}{s}]$
LQR	1	0.6200	6.5841	14.0997	3.2550
LQR	2	0.7700	7.8375	14.1007	3.5919
LQR	3	0.5700	6.1508	14.1100	3.2396
LQR	4	0.7500	8.4301	14.1153	4.0949
LQR	5	0.6900	7.5912	14.0896	3.3436
LQR	6	1.0900	9.3344	14.1649	4.1689
LQR	7	1.0100	8.6831	14.0772	3.2496
LQR	8	1.0200	7.5075	14.0404	3.0553
LQR	9	0.6700	8.5227	14.0719	3.2367
LQR	10	0.7300	7.7260	14.1257	4.6143
PPO	1	0.7500	4.6639	14.0732	3.3405
PPO	2	0.6700	4.3294	14.0649	3.3147
PPO	3	0.7700	4.8435	14.1540	4.5967
PPO	4	0.9600	4.7568	14.1117	5.3525
PPO	5	0.6900	4.7445	14.1163	3.0337
PPO	6	0.7200	4.4122	14.1573	5.2125
PPO	7	0.8300	5.3388	14.1207	4.8015
PPO	8	0.9400	3.2477	14.0671	3.7742
PPO	9	1.1100	4.1034	14.1436	4.0622
PPO	10	0.8900	3.9165	14.1493	3.5671
LQR	**Avg**	**0.7920**	**7.8367**	**14.0996**	**3.5850**
PPO	**Avg**	**0.8330**	**4.4357**	**14.1158**	**4.1056**

C.3 Test Results for Model Error Test

Table C.7: Test results using LQR from model error test $\hat{\varphi}_0 = 9\ [deg]$.

Test #	Stabilized	μ_φ	$\varphi_0\ [deg]$	$\dot{\varphi}_0\ [\frac{deg}{s}]$	$\theta_0\ [deg]$	$\dot{\theta}_0\ [\frac{deg}{s}]$
1	Yes	2.2935	9.0943	3.0210	-0.0469	-0.3736
2	Yes	0.86413	9.0893	3.1237	-0.1207	0.2772
3	Yes	1.3702	9.0472	3.2381	-0.1394	-0.9110
4	No	-	9.0821	3.6097	-0.4104	-1.5084
5	Yes	1.5176	9.0856	3.0243	-0.3172	0.3485
6	Yes	1.0713	9.1114	3.4831	-0.0155	0.6072
7	Yes	2.472	9.0152	3.6989	-0.1006	1.6323
8	Yes	1.5314	9.1662	3.7062	-0.7624	-3.2506
9	Yes	1.251	9.0478	4.1718	0.2071	1.3157
10	Yes	1.1035	9.1073	3.1890	0.0548	0.0839

Table C.8: Test results using PPO from model error test $\hat{\varphi}_0 = 9\ [deg]$.

Test #	Stabilized	μ_φ	$\varphi_0\ [deg]$	$\dot{\varphi}_0\ [\frac{deg}{s}]$	$\theta_0\ [deg]$	$\dot{\theta}_0\ [\frac{deg}{s}]$
1	Yes	1.2	9.0753	3.0410	-0.0753	-0.3613
2	Yes	1.2806	9.0999	3.8043	-0.2346	-0.5411
3	Yes	1.1943	9.1434	3.5626	-0.0405	-0.6100
4	Yes	1.1814	9.1042	3.7209	-0.0384	-0.7838
5	Yes	1.9879	9.0939	3.6663	0.0476	0.8471
6	Yes	-0.367	9.0892	3.2021	0.0471	0.4288
7	No	-	9.0259	4.1683	-0.1830	0.2996
8	No	-	8.9996	4.6991	-0.3876	-3.4415
9	Yes	1.9286	9.0412	4.5242	-0.1508	-0.8677
10	No	-	9.1138	3.3368	-0.1250	0.7790

Table C.9: Test results using LQR from model error test $\hat{\theta}_0 = 29\ [deg]$.

Test #	Stabilized	$\theta_0\ [deg]$	$\dot{\theta}_0\ [\frac{deg}{s}]$	$\varphi_0\ [deg]$	$\dot{\varphi}_0\ [\frac{deg}{s}]$
1	Yes	28.8810	3.7196	-0.4086	-0.5598
2	Yes	28.9892	3.4049	0.0674	0.8184
3	Yes	29.0633	5.6467	-0.4138	-0.1552
4	Yes	28.9391	3.7552	-0.0098	1.2048
5	No	29.1707	4.4005	-0.0448	0.0508
6	Yes	28.9746	3.7016	-0.0908	-0.2888
7	Yes	29.0673	5.1658	-0.0678	-1.6815
8	No	29.0048	3.7917	-0.1430	-1.5184
9	Yes	29.0150	3.1857	-0.0494	-2.0361
10	Yes	29.2392	3.0602	-0.1964	-4.6488

Table C.10: Test results using PPO from model error test $\hat{\theta}_0 = 21\ [deg]$.

Test #	Stabilized	$\theta_0\ [deg]$	$\dot{\theta}_0\ [\frac{deg}{s}]$	$\varphi_0\ [deg]$	$\dot{\varphi}_0\ [\frac{deg}{s}]$
1	Yes	21.0552	3.6683	-0.3007	-0.2846
2	No	21.0940	3.7070	-0.1747	-1.9587
3	Yes	21.0733	3.1835	-0.2771	-1.5523
4	Yes	21.0543	4.1781	-0.1358	0.3921
5	No	20.9660	4.5332	-0.0790	-2.1191
6	No	20.9233	4.4010	0.0917	-2.6484
7	No	21.1447	3.4218	-0.0944	1.0080
8	Yes	21.0218	4.0549	-0.1044	-1.2701
9	Yes	21.0041	3.4832	-0.1214	0.3221
10	No	21.0570	4.0286	-0.5909	-1.7401

C.4 Test Results for External Impulse Test

Table C.11: Test results applying external impulse on the LQR

Test #	Stabilized	θ_0 $[deg]$	$\dot{\theta}_0$ $[\frac{deg}{s}]$	φ_0 $[deg]$	$\dot{\varphi}_0$ $[\frac{deg}{s}]$
1	Yes	0.5557	6.5918	-0.0876	17.1418
2	No	0.8183	7.0947	0.4986	19.9469
3	Yes	0.1359	-9.2007	0.1313	15.1581
4	Yes	0.3978	-0.0542	-0.4335	-3.1423
5	No	0.2645	-3.3012	0.2896	18.2692
6	Yes	0.2207	-0.4678	0.1535	13.4951
7	No	0.3442	0.1493	-0.8513	-3.0077
8	Yes	0.2534	-0.3308	-0.1437	16.2608
9	Yes	0.4412	0.3076	0.3329	17.4939
10	No	0.3909	8.3626	0.4904	21.6636

Table C.12: Test results applying external impulse on the PPO

Test #	Stabilized	θ_0 $[deg]$	$\dot{\theta}_0$ $[\frac{deg}{s}]$	φ_0 $[deg]$	$\dot{\varphi}_0$ $[\frac{deg}{s}]$
1	Yes	-0.3992	-0.8181	0.4079	-3.1005
2	Yes	0.7291	8.5475	0.2601	19.8193
3	Yes	-0.2359	7.7453	0.123526	19.6751
4	Yes	0.1865	0.9738	-0.3499	-3.0710
5	No	0.7351	1.95763	-0.2147	8.5414
6	Yes	-0.1593	-1.7963	-0.5022	8.4269
7	Yes	0.4803	1.4596	0.2353	14.4134
8	Yes	0.4306	-2.6988	-0.0455	11.2114
9	No	0.7027	3.1732	0.1334	16.7592
10	No	0.2180	-0.7331	0.2663	15.3498