Self Stabilizing Still-Standing Bike

Balancing a still-standing bike using a gyro

Bachelor's thesis in Department of Electrical Engineering

GUSTAF ANDERSSON, FELIX DREJMO
OLIVER KRYGER, MOHAMAD ISAK OBAID
NILAS VASUR, ALEXANDER WERMLUND

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Bachelor's Thesis 2026

Self Stabilizing Still-Standing Bike

Balancing a still-standing bike using a gyro

GUSTAF ANDERSSON
FELIX DREJMO
OLIVER KRYGER
MOHAMAD ISAK OBAID
NILAS VASUR
ALEXANDER WERMLUND

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2026

Self Stabilizing Still-Standing Bike
Balancing a still-standing bike using a gyro
GUSTAF ANDERSSON, FELIX DREJMO, OLIVER KRYGER, MOHAMAD ISAK OBAID, NILAS VASUR, ALEXANDER WERMLUND

© GUSTAF ANDERSSON, FELIX DREJMO, OLIVER KRYGER, MOHAMAD ISAK OBAID, NILAS VASUR, ALEXANDER WERMLUND, 2026.

Supervisor: Qianlei Jia, Department of Electrical Engineering
Examiner: Jonas Sjöberg, Department of Electrical Engineering

Bachelor's Thesis 2026
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: Photo of the black bike with implemented solution with the gyro balancing the stationary bike.

Typeset in LaTeX
Gothenburg, Sweden 2026

Self Stabilizing Still-Standing Bike
Balancing a still-standing bike using a gyro
GUSTAF ANDERSSON, FELIX DREJMO, OLIVER KRYGER
MOHAMAD ISAK OBAID, NILAS VASUR, ALEXANDER WERMLUND
DEPARTMENT OF ELECTRICAL ENGINEERING
Chalmers University of Technology

Abstract

This report presents the development of a control system for balancing a stationary bicycle. The system uses a gyro driven by a brushless motor to apply a stabilizing torque, enabling balance without any forward velocity. The project models the bicycle as an inverted pendulum, described using a nonlinear state-space representation. Roll angle estimation is achieved through a complementary filter fusing accelerometer and gyroscope data from an IMU. A cascade PID control structure regulates the motor current in response to the measured tilt. The control algorithm was developed for implementation on a National Instruments MyRIO platform using LabVIEW and then implemented on an ESP32 microcontroller as a standalone alternative due to hardware and software compatibility issues. Simulation results demonstrate that the controller is capable of stabilizing the bicycle within the physical limits of the hardware. Hardware and software compatibility issues between different bicycles in the Autobike project limited full experimental validation, and several directions for future development are identified and discussed.

Keywords: Autobike, Self Balancing Bicycle, LabVIEW, Arduino, C, Control theory, Sensor Fusion, PID, MyRIO, Gyro.

Acknowledgements

This thesis is the result of a collaboration between students with different academic backgrounds. The areas of study are Mechanical, Mechatronics and Computer Engineering during the spring of 2026 at Chalmers University of Technology.

We would like to thank our examiner Jonas Sjöberg for the opportunity to work on this project. We would also like to thank our supervisor Qianlei Jia for the guidance and help provided during the project with LabVIEW and other assistance with implementing a solution. One more person we would like to thank is Seyedemadaddin Ashrafmansouri, that despite not having any formal responsibility with our project, helped us in the laboratory.

Gustaf Andersson, Felix Drejmo, Oliver Kryger,
Mohamad Isak Obaid, Nilas Vasur and Alexander Wermlund
Gothenburg, June 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

CLF	Call Library Function
IMU	Inertial Measurement Unit
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
NI	National Instruments
SO	Shared Objects
VESC	Vedder Electronic Speed Controller
VI	Virtual instrument

Nomenclature

Below is the nomenclature of parameters, and variables that have been used throughout this thesis.

Parameters

m_{tot}	Total mass of the bicycle
g	Gravitational constant
l_c	Distance from pivot point to the bicycle's center of mass
J_{tot}	Total moment of inertia of the bicycle
J_g	Moment of inertia of the gyroscope
R	Electrical resistance in the motor
L	Electrical inductance in the motor
b	Motor friction
K_m	Motor constant
K_v	Motor velocity constant
I_{max}	Motor current tolerance
V_{nom}	Battery nominal voltage
dt	Time step constant

Variables

ϕ	Calculated roll angle with complementary filter
ϕ_{prev}	Previous calculated roll angle from complementary filter
ϕ_{gyro}	Calculated roll angle with gyroscope from IMU
ϕ_{acc}	Calculated roll angle with accelerometer from IMU
ω_g	Angular velocity of the gyro
$\dot{\omega}_g$	Angular acceleration of the gyro

ω_{imu}	Measured angular velocity with IMU
a_{imuY}	Measured acceleration in cartesian Y-axis by the IMU
a_{imuZ}	Measured acceleration in cartesian Z-axis by the IMU
α	Weighing factor of complementary filter
u	Supplied voltage to the motor
i	Supplied current to the motor

Contents

List of Acronyms **vi**

Nomenclature **vi**

List of Figures **xi**

List of Tables **xii**

1 Introduction **1**
- 1.1 Purpose . . . 1
- 1.2 Goals . . . 1
- 1.3 Limitations . . . 2
- 1.4 Usage of AI . . . 2

2 Background **3**
- 2.1 Different bicycles . . . 3
- 2.2 Hardware . . . 4
  - 2.2.1 Embedded system . . . 4
  - 2.2.2 Alternative embedded system (ESP32) . . . 5
  - 2.2.3 Motor-unit and gyro . . . 6
  - 2.2.4 VESC . . . 6
  - 2.2.5 IMU . . . 7
  - 2.2.6 Battery . . . 7
- 2.3 Software . . . 8
  - 2.3.1 Programming languages . . . 8
  - 2.3.2 LabVIEW . . . 8
  - 2.3.3 Arduino IDE . . . 9

3 Theory **10**
- 3.1 Modeling of physical aspects of the system . . . 10
- 3.2 Modeling of the electrical aspects of the system . . . 11
- 3.3 State space model . . . 12
- 3.4 Hardware limitations . . . 12
  - 3.4.1 Motor Angular Velocity Limit . . . 12
  - 3.4.2 Maximum current . . . 13
  - 3.4.3 Angle offset limit . . . 13

4 Method **14**
4.1 Roll angle calculation 14
4.1.1 Complementary filter in LabVIEW and C-code 15
4.2 Control loop 16
4.2.1 Current control in LabVIEW 17
4.2.2 Motor control in LabVIEW 18
4.3 Full SIMULINK simulation with limitations 18
4.3.1 Process - Subsystem 19
4.3.2 Angle estimation - Subsystem 20
4.3.3 Tuning 20
4.4 Standalone balancing control with an ESP32 21
4.5 Mount for motor on the black bike 21

5 Results **22**
5.1 Simulation Results 22
5.1.1 Time Varying Errors 22
5.1.2 Initial Offset Error 23
5.1.3 Gyro bias 25
5.2 Performance of Complementary filter 26
5.3 Balancing Results 26

6 Conclusion **27**
6.1 Discussion 27
6.2 Simulation vs. Real System 27
6.3 Further development 28

A Code **I**
A.1 Simulations I
A.2 Roll angle calculation, complementary filter III
A.3 Control signal calculation IV
A.4 Implementation in ESP32 V

List of Figures

2.1 The orange bike used as concept in the project 3
2.2 The black bike used to implement solution 4
2.3 MyRIO hardware 5
2.4 Early version of PCB-board used as interface 5
2.5 ESP32, used in alternative standalone system 5
2.6 ESP32 in system connected to IMU and VESC 5
2.7 Motor and gyro assembly used to balance the bicycle 6
2.8 VESC unit to control the balancing motor 6
2.9 Digilent Pmod NAV, IMU mounted in the black bicycle 7
2.10 Li-poly battery used in the project to test outside 7
2.11 To the left, showcase the general structure of a loop which runs a sub-VI in the main-VI and to the right, the general structure how to run C-code with a CLF-node in a sub-VI 9

3.1 A model of the bicycle as an inverted pendulum with the gyroscope connected 11
3.2 Electrical circuit of the DC-motor 11

4.1 Visualization of parameters used to calculate the roll angle 15
4.2 LabVIEW main- and sub-VI of the implementation of the complementary filter 16
4.3 Cascade control loop of the system 17
4.4 LabVIEW main- and sub-VI of the current control loop 17
4.5 LabVIEW main VI motor control function 18
4.6 Non-linear simulation model in Simulink 19
4.7 Process subsystem, simulated model of the bicycle 19
4.8 Angle estimation subsystem, simulated model of the complementary filter 20
4.9 CAD model of bike-motor attachment 21

5.1 Simulink simulation showing how the roll angle changes with an initial 1.5° offset under time-varying accelerometer and gyroscope noise. . . 23
5.2 Simulink simulation showing how the current and voltage signals changes with an initial 1.5° offset under time-varying accelerometer and gyroscope noise. 23

5.3 Simulink simulation showing how the roll angle changes with a 1.46° initial offset when the IMU starts with a constant measurement error of 0.2°. 24
5.4 Simulink simulation showing how the current and voltage signals changes with a 1.46° initial offset when the IMU starts with a constant measurement error of 0.2°. 24
5.5 Simulink simulation showing how the roll angle changes with a 1.51° initial offset when the IMU is affected by a constant gyro bias of $0.2°/s$. 25
5.6 Simulink simulation showing how the current and voltage signals changes with a 1.51° initial offset when the IMU is affected by a constant gyro bias of $0.2°/s$. 25
5.7 Complementary filter with different values on α 26

List of Tables

3.1 Assumed values of parameters used in calculations 10
3.2 Motor parameters (AXI 5360/24HD V3 GOLD LINE) 12
3.3 Battery parameters (Turnigy High Capacity 6S 14000mAh 22.2V 12C Lipo Battery w/XT90) 12
3.4 Showcasing the correlation between roll angle, torque and current . . 13

1 Introduction

The development of autonomous and self-stabilizing bicycles has gained interest in traffic safety research. At Chalmers University of Technology and Mälardales University (MDU), a series of self-driving bicycles have been developed to be used in controlled experiments to evaluate vehicle safety systems. These bicycles mimic real cyclists instead of relying on track mounted bicycles, creating a wider range of possible tests which are more analogous to real scenarios. In current implementations, the bicycles rely on steering control to maintain balance during motion. This approach exploits the natural stability that arises when a bicycle is moving forward. However, such systems require a minimum velocity to remain stable, which limits their scope to when the bike is in motion.

1.1 Purpose

The purpose of this project is to investigate and demonstrate the feasibility of stabilizing a stationary bicycle using a controlled gyroscopic system. It aims to develop and implement a control system to maintain an upright position without relying on forward motion. This will also require an estimation algorithm with sensor data from an Inertial Measurement Unit (IMU) to obtain a reliable estimate of the bicycle's roll angle.

1.2 Goals

The goal of this project is to implement a functional solution in which the bike can autonomously balance without human interaction for extended periods of time and correct smaller deviations relative to its upright equilibrium. The aim is to demonstrate that the system can be stabilized and that the roll angle can be effectively controlled using sensor data and control algorithms. To achieve these goals, the project is divided into the following subgoals:

- Development of a mathematical model of the entire system.
- Development of a simple estimation algorithm for the roll angle based on IMU measurements.
- Implementation of the mathematical model in Simulink, verifying roll angle limitations based on sensor errors and hardware limitations.
- Implementation of the control algorithm in the existing system in C and LABVIEW or similar software solution.
- Performing validation tests on the physical system.

1.3 Limitations

The limitations that have been realized during the project are primarily related to the preexisting hardware and software. One of the limitations is a direct consequence of the objective, that the analysis and development is limited to the case where the bike is stationary and the transition between stationary and forward motion has not been taken into consideration. Another limitation is the amount of current available for use; both in the case of the power supply in the lab and the battery. There is also a limitation realized during the project with the existing software given in combination with the amount of changes we were allowed to make to the hardware. This has had two major consequences. The first of which being that the bike was changed during the project. When that changed, the hardware and software did not align and the amount of troubleshooting necessary to run was not finalized before the deadline.

1.4 Usage of AI

During the development of the still standing bicycle, different AI-models have been used with a well-thought-out motive and questions to get as good of an answer as possible. The answers provided by the different AI models have been critically evaluated and verified against other sources. This has made the process in many cases quicker since the first information is given and then the details have been checked. The different AI-models used differ between tasks like research, code identification in LabVIEW and code development. The models used most in the project are Google Gemini, OpenAI ChatGPT and Microsoft Copilot. The majority of usage of AI is related to checking code for bugs and checking text for grammar and spelling mistakes. It has also been used to speed up the development process since the code editor mostly used, Visual Studio Code, has an integration with Copilot where it gives suggestions related to what it thinks the code is supposed to accomplish.

2
Background

Much of the hardware and software used in this project has been developed and used previously during the Autobike project, a collaboration between the Department of Electrical Engineering at Chalmers and MDU. This section provides an overview of the technical foundation for this project and describes the relevant parts from the Autobike project.

2.1 Different bicycles

The overarching project consists of multiple different bicycles and an electric scooter. The bikes are usually named by their corresponding main or accent color and developed with similar hardware and software but with individual solutions to balance. The bike which this project was set to focus on was the orange bike, Figure 2.1. The first half of the project was done having this bike in mind.

Figure 2.1: The orange bike used as concept in the project

Due to the Autobike project being worked on in parallel to this one, checking and changing connections on the hardware was not possible. Thus, the solution was instead implemented on the black bike Figure 2.2. The black bike was developed early in the Autobike project, which led to issues with software and hardware compatibility. Solving these problems late into the project was not feasible; and thus the scope of the solution was reduced in order to get a working implementation within the given timeframe.

Figure 2.2: The black bike used to implement solution

2.2 Hardware

The basic hardware is the same among all bicycles. In this project, some of the hardware and software is necessary for the understanding which will be described below. There are also some parts of the system that are not taken into consideration in this project, like the motor used for steering and the system, stabilizing when the bike is in motion. For a broader explanation of the system as a whole, we refer to three of the previous reports in the Autobike project [1][2][3].

2.2.1 Embedded system

The embedded system used in the Autobike project is the MyRIO-1900 created by National Instruments (NI) and is used as the central hub, where all of the calculations are made [4]. The MyRIO is connected to all of the other systems and information is sent between every part of the system. It is also connected to a specific PCB-board which acts as an interface to make connectors to all of the systems and connect them to the right pins on the MyRIO. This interface has changed which created compatibility issues with the current Autobike software. Below in Figure 2.4 the first version of the PCB is presented and in the report [5] more information about the newer version can be found.

Figure 2.3: MyRIO hardware

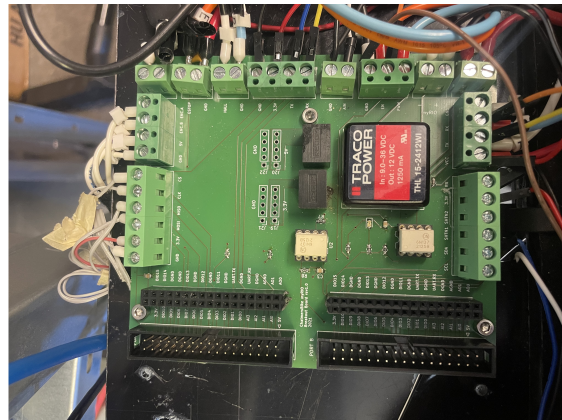


Figure 2.4: Early version of PCB-board used as interface

2.2.2 Alternative embedded system (ESP32)

As a result to the difficulties with the existing MyRIO-based implementation, a decision was made to implement a stand-alone balancing controller using an alternative embedded platform, the ESP32 developed by Espressif Systems. The ESP32 is supported by Arduino which allows the microcontroller to be programmed using the Arduino IDE and the C++ programming language. This allows the control algorithm, including the complementary filter and PID controller, to be written and deployed directly without the need for a separate development environment or compiled shared-object files, as required by the MyRIO platform.

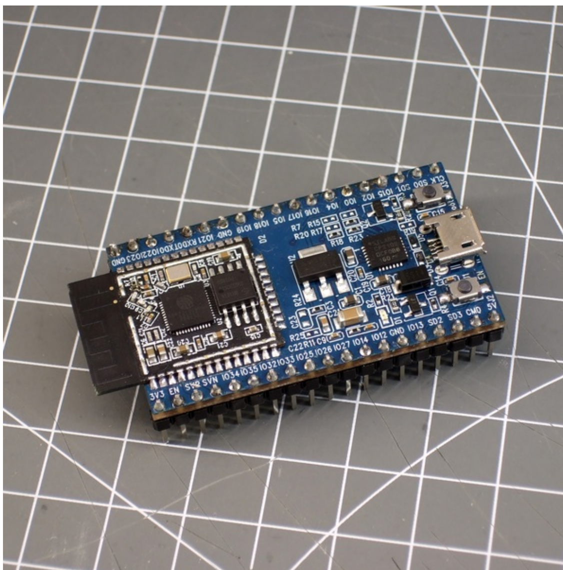


Figure 2.5: ESP32, used in alternative standalone system

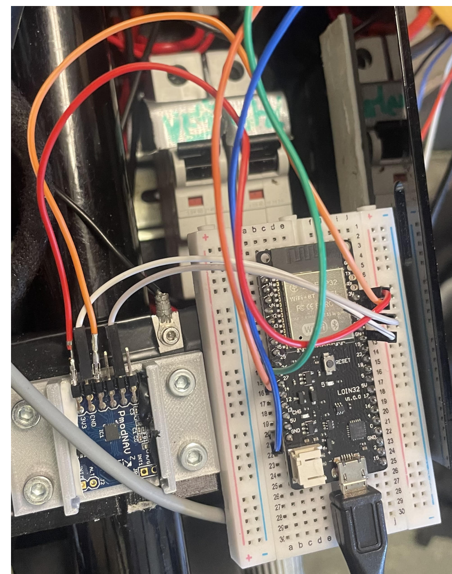


Figure 2.6: ESP32 in system connected to IMU and VESC

2.2.3 Motor-unit and gyro

To be able to balance the bicycle, earlier projects have developed a concept of having a motor, connected to a gyro which axis of rotation is in the direction of travel. The motor that creates the torque is a brushless, 3 phase, motor which with the help of a speed controller can control the speed of the motor. The motor used in the system is AXI 5360/24HD V3 GOLD LINE [6].

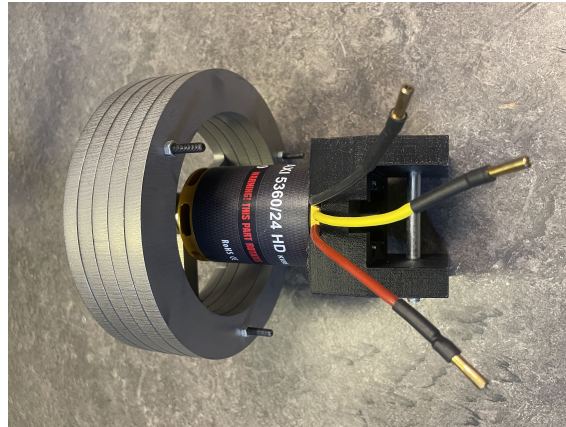


Figure 2.7: Motor and gyro assembly used to balance the bicycle

2.2.4 VESC

VESC is an Electric Speed Controller (ESC) which has been used earlier in the Autobike project to control the speed of different motors. In this project it is used to control the balancing motor by sending a current from the voltage source. It also allows the solution to set limits of how much current and voltage it will give, enhancing safety for the battery, ESC and motor. The model used in this project is the VESC 6 MK VI [7] [8].

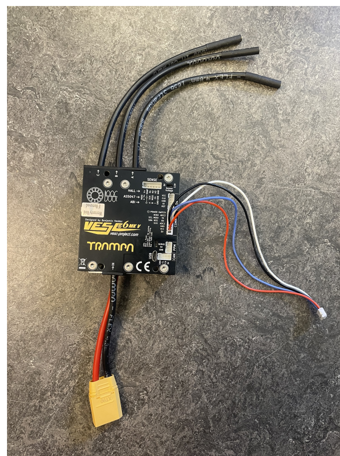


Figure 2.8: VESC unit to control the balancing motor

2.2.5 IMU

An IMU is a sensor system capable of measuring how an object moves and rotates in a 3D space. The IMU used in the system is the Digilent Pmod NAV [9]. It consists of an accelerometer which measures linear acceleration along it's 3 cartesian axes, as well as a gyroscope that measures angular velocity around the same 3 axes. The data is discrete with a chosen sampling interval, and will be used to calculate the roll angle of the bike.

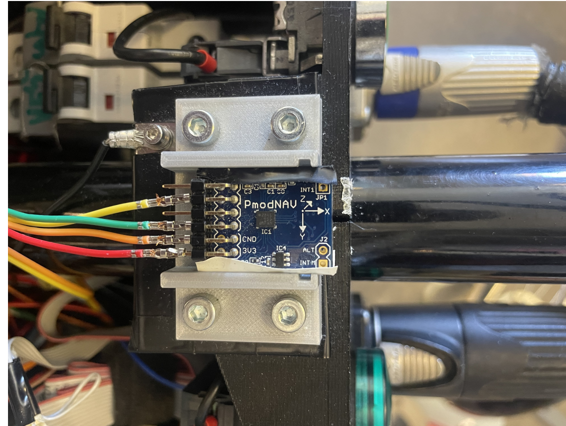


Figure 2.9: Digilent Pmod NAV, IMU mounted in the black bicycle

2.2.6 Battery

The battery used is the "Turnigy High Capacity 6S 14000mAh 22.2V 12C Lipo Battery w/XT90" [10]. This is a 6s Li-Poly battery with a nominal voltage (V_{nom}) of 22.2 V. It has a discharge rate of 12 C and a capacity of 14 Ah making it capable of discharging 168 A.

Figure 2.10: Li-poly battery used in the project to test outside

2.3 Software

The first half of this project used the same software architecture as the Autobike project (LabView and C), further developing the existing software. In order to perform physical testing, our work was later implemented in Arduino IDE using C++. This section provides background for all the software that has been used in this project.

2.3.1 Programming languages

Four different languages have been utilized during the project. MatLab in collaboration with Simulink to simulate the system, python to create models and simulations of parts of the system and C and C++ to implement functionality from LabVIEW and Arduino onto the real bicycle.

2.3.2 LabVIEW

Laboratory Virtual Instrument Engineering Workbench (LabVIEW) is the programming backbone of the Autobike project and it is required for the MyRIO-1900 to run [11]. The software is a visually based programming language which is used throughout the industry. The environment LabVIEW runs the language G, that is the graphical structured language NI created, and it runs Virtual Instruments (VIs) which is constructed via two different interfaces. The front panel which is the interface where the user can interact with the program when it runs, and second, block diagram which is where the user programs the functionality of the program [12].

In the Autobike project, LabVIEW is used extensively, and since all of the bikes have the same base hardware the basic structure of the program is the same and can run on every bike. In order to run the program there are 3 necessary files. Calibration, configuration and main and these files need to be run consecutively after connecting to the bicycle. The first two files are executed once at startup, while the main file runs continuously, unless manually stopped. The main file will continuously run other sub VIs inside loops which control specific aspects that the bicycle needs to function. The majority of these sub VIs run a C-file that has been converted into a shared objects (.so) -file which is a compiled library file that a Linux system can run [13]. This file runs in LabVIEW through a Call Library Function node (CLF-node) which takes inputs and calculates outputs to be used by other parts of the system. An example of this is shown below in Figure 2.11 where the main file runs a loop of the sub-VI and then the sub-VI takes the inputs into the CLF-node and calculates the output that it sends to the main-VI.

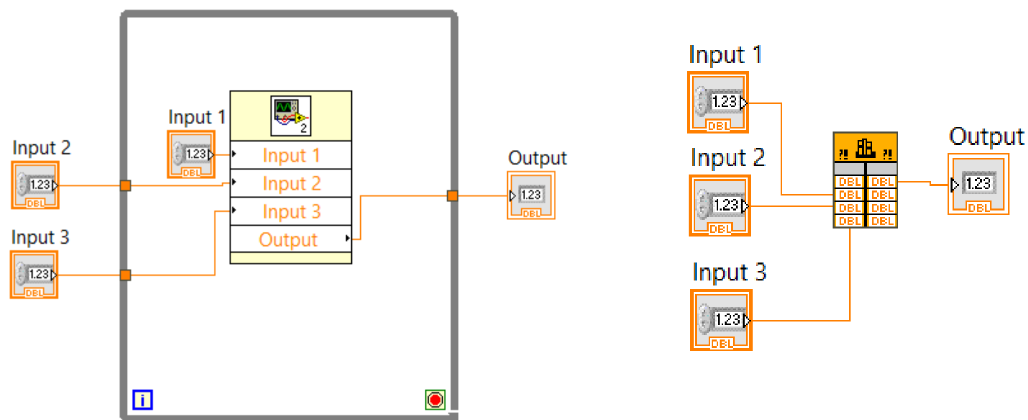


Figure 2.11: To the left, showcase the general structure of a loop which runs a sub-VI in the main-VI and to the right, the general structure how to run C-code with a CLF-node in a sub-VI

2.3.3 Arduino IDE

The Arduino IDE is a development environment used to program the ESP32 in this project. It supports the C++ programming language and libraries needed, such as SparkFun LSM9DS1 and VescUart library to control the external hardware used in this project with the IMU and VESC. The environment allows code to be written, compiled and flashed directly to the microcontroller through a serial connection, without the need for additional build tools or a runtime environment.

3
Theory

This chapter aims to describe the prerequisite theory and modeling aspects used in the project. To perform calculations, parameter values were estimated based on a series of assumptions, these are summarized in Table 3.1.

Parameter	**Value**
m_{tot}	40 kg
l_c	0.5 m
J_{tot}	10 kgm^2
J_g	0.1575 kgm^2
L	3 mH

Table 3.1: Assumed values of parameters used in calculations

3.1 Modeling of physical aspects of the system

The roll dynamics of the bicycle can be described using a simplified model, illustrated below in Figure 3.1. That is, as an inverted pendulum with one contact point to the ground.

The model approximates the bicycle as a point-mass which as a function of its roll angle (ϕ) and gravitational acceleration, creates a moment perpendicular to the point of contact. This moment is then counteracted by the motor and gyro combination which creates the moment on the bicycle where the motor is mounted, which is proportional and opposite to the angular torque that the accelerating gyro creates. This is all according to Newtons second law of rotation together with Newtons third law and described as:

$$J_{tot}\ddot{\phi} = m_{tot}l_c g sin(\phi) - J_g\dot{\omega}_g \tag{3.1}$$

By taking the moments about the center of the gyro and applying the principle for an electric motor $\tau_m = K_m i(t)$, the dynamic relationship between the gyro-speed (ω_g) and the current ($i(t)$) is obtained. There is also a friction that is affected by the gyro's angular speed which is subtracted. By using Newton's second law for rotation, the equation is constructed as follows:

$$J_g\dot{\omega}_g = K_m i(t) - b\omega_g \tag{3.2}$$

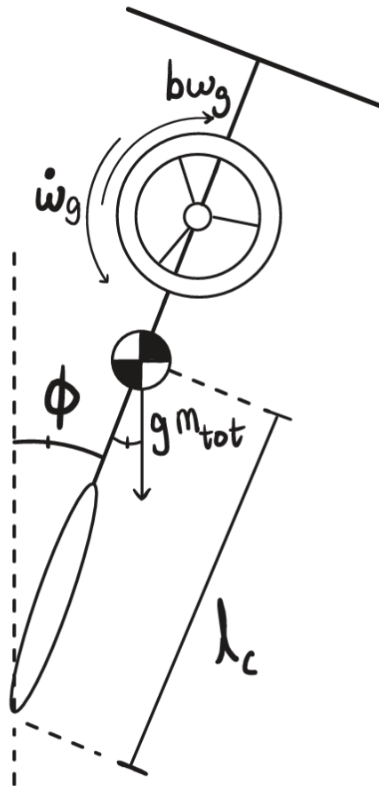


Figure 3.1: A model of the bicycle as an inverted pendulum with the gyroscope connected

3.2 Modeling of the electrical aspects of the system

The gyro is connected to a DC motor, the circuitry of which is described in figure 3.2. Applying Kirchhoff's Voltage Law to the circuit, together with the relation describing the voltage drop across the motor, $u_m = K_u \omega_g$ yields equation:

$$u(t) = Ri(t) + L\dot{i}(t) + K_u \omega_g \tag{3.3}$$

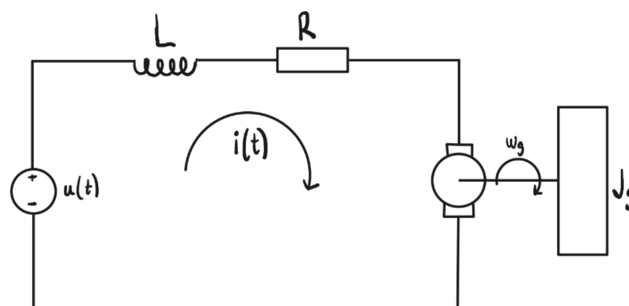


Figure 3.2: Electrical circuit of the DC-motor

3.3 State space model

By combining equation 3.1, 3.2 and 3.3, a nonlinear state-space representation of the system can be formulated as follows:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} \phi \\ \dot{\phi} \\ \omega_g \\ i \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{1}{J_{tot}}\Big(m_{tot} g l_c \sin(x_1) - K_m x_4 + b x_3\Big) \\ \frac{1}{J_g}\Big(K_m x_4 - b x_3\Big) \\ \frac{1}{L}\Big(u(t) - R x_4 - K_u x_3\Big) \end{bmatrix}$$

3.4 Hardware limitations

The system's ability to balance requires a torque generated by the motor and the reaction wheel, this torque needs to be larger than the gravitational torque that stems from the roll angle. This section aims to provide guidelines for the range of roll angles that are possible to correct by the motor used in this project. Table 3.2 and 3.3 below shows the motor [6] and battery [10] parameters used for calculations.

Parameter	**Value**
Rated number of cells	12s Li-Poly
Motor velocity constant (K_v)	95 rpm/V
Current tolerance (I_{max})	65 A / 20 s
Internal resistance (R)	63 mΩ
Torque constant (K_m)	0.1 Nm/A

Table 3.2: Motor parameters (AXI 5360/24HD V3 GOLD LINE)

Parameter	**Value**
Number of cells	6s Li-Poly
Nominal voltage (V_{nom})	22.2 V
Discharge rate	12 C
Capacity	14 Ah

Table 3.3: Battery parameters (Turnigy High Capacity 6S 14000mAh 22.2V 12C Lipo Battery w/XT90)

3.4.1 Motor Angular Velocity Limit

The motor is designed for use with a 12s Li-Poly battery, however the one used in this project is a 6s Li-Poly which limits the maximum nominal voltage (V_{nom}) to 22.2 V. The theoretical maximum angular velocity (ω_{max}) is determined by the motor velocity constant K_v and the applied voltage according to:

$$\omega_{g,max} = \frac{K_v \cdot V_{nom} \cdot 2\pi}{60} \tag{3.4}$$

With a specified K_v of 95 RPM/V and a nominal voltage of 22.2 V, which results in a maximum angular velocity of:

$$\omega_{g,max} = \frac{95 \cdot 22.2 \cdot 2\pi}{60} \approx 221\text{rad/s} \tag{3.5}$$

If the motor reaches this limit, it is saturated and can no longer provide the angular acceleration $\dot{\omega}_g$ needed to balance.

3.4.2 Maximum current

The motor has a current capacity of 65 A / 20 s. The battery is capable of discharging up to 168 A, making the motor the limiting component when accelerating. When the system is decelerating however, the battery is using regenerative braking for a limited time. Given that the recharge time is limited, we assumed that 3 C (three times the battery capacity rating of 14 Ah) is a reasonable limit, making it able to decelerate the motor with a current of 42 A. It's important to point out that it's difficult to properly know the safe limits for usage like this, but it's clear that the battery is the bottleneck when the motor is decelerating.

3.4.3 Angle offset limit

In a stationary position, an angular acceleration ($\ddot{\phi}$) is required for the bicycle to move toward equilibrium. According to equation 3.1, this implies that the torque from the motor must be greater than the torque exerted by gravity, where the gravitational torque is defined as:

$$\tau_g = m_{tot} g l_c \cdot sin(\phi) \tag{3.6}$$

Assuming negligible friction in Equation 3.2, the motor torque is defined as:

$$\tau_m = J_g \dot{\omega}_g = K_m i(t) \tag{3.7}$$

Combining equation 3.4 and 3.5 provides an isolated expression for the required current at an arbitrary angle:

$$i(t) > \frac{1}{K_m} m_{tot} g l_c sin(\phi) \tag{3.8}$$

Angle [deg]	**Torque [Nm]**	**Current [A]**
1	3.43	34.3
2	6.85	68.5
3	10.28	102.8

Table 3.4: Showcasing the correlation between roll angle, torque and current

4 Method

This section describes the developments of hardware and software during the project. Showing how the bicycle is set up to calculate the appropriate control signals for the still standing bicycle with the gyro.

4.1 Roll angle calculation

Determining an objects position using data from an accelerometer and a gyroscope is possible through a method called sensor fusion. As the name suggests, this method combines multiple sensor inputs to produce a more reliable estimate. While sensor fusion has many applications, this project specifically focuses on calculating the bike's roll angle [14].

The information provided by the IMU is the linear acceleration in all directions of the cartesian coordinate system and the angular rotation rate around each axis [15]. The data is collected and used to calculate the angle with the previous roll angle in consideration. The variables, with descriptions, used in the equations below to calculate the roll angle, can be found in the nomenclature.

The signals from the accelerometer and gyroscope have different strengths and weaknesses. The gyroscope is a stable signal that senses very small changes, but it has a major flaw, which is that it drifts over time. This means that over time the calculation has a large error and the approximation of the angle becomes unreliable. The calculation of the angle from the gyroscope is calculated by taking the previous roll angle and adding the angular velocity in the direction of roll multiplied by the time period, according to:

$$\phi_{\text{gyro}} = \phi_{\text{prev}} + \omega_{\text{imu}} * dt \tag{4.1}$$

The accelerometer does not drift like the gyroscope but the value calculated oscillates around a center point. The roll angle can then be calculated using trigonometry with the gravitational acceleration sensed by the accelerometer in the cartesian Y and Z direction, according to:

$$\phi_{\text{acc}} = tan^{-1}\frac{a_{\text{imuY}}}{a_{\text{imuZ}}} \tag{4.2}$$

Equation 4.1 and 4.2 is then added together with a weighting, α, which highlights

strengths and eliminates weaknesses from the sensors. A small part of the accelerometer, around 1-3% and the rest is taken from the gyroscope, described below:

$$\phi = \alpha * \phi_{\text{gyro}} + (1.0 - \alpha) * \phi_{\text{acc}} \tag{4.3}$$

Figure 4.1 below, shows a summary of all the accelerations and the roll rate used to calculate the roll angle.

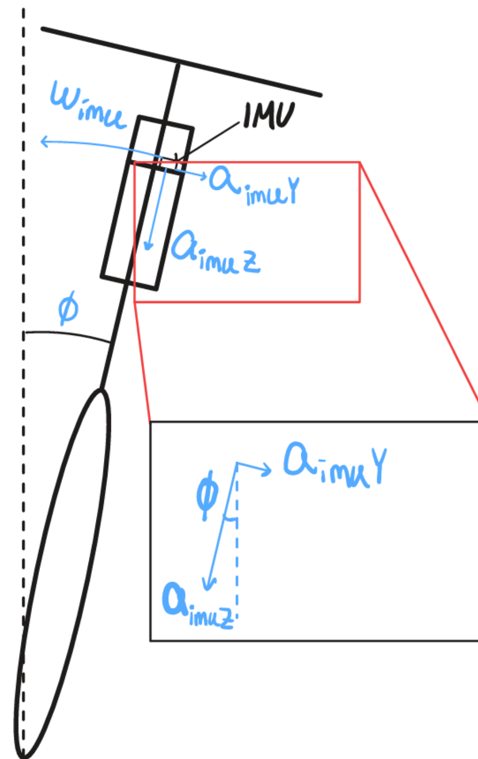


Figure 4.1: Visualization of parameters used to calculate the roll angle

4.1.1 Complementary filter in LabVIEW and C-code

The implemented version of the complementary filter is found in appendix A.3. This code is then turned into an .so file, which stands for shared object and is a binary file which the MyRIO hardware can read. The code is put into a call library function node which in its turn takes input from the system, which is connected in LabVIEW, and produces the values that the C function provides. This is shown in Figure 4.2. Figure 4.2 shows how the filter is implemented in the main LabVIEW VI where an express VI module is running the sub-VI of the complementary filter. It takes input of measurements from other sub-VIs and the set values for the gyro amount and gyro threshold to create a new estimation of the roll angle. The roll angle value from the sub-VI is then connected through a shift register back to the input of the roll angle and the while loop which contains the functionality runs for another iteration.

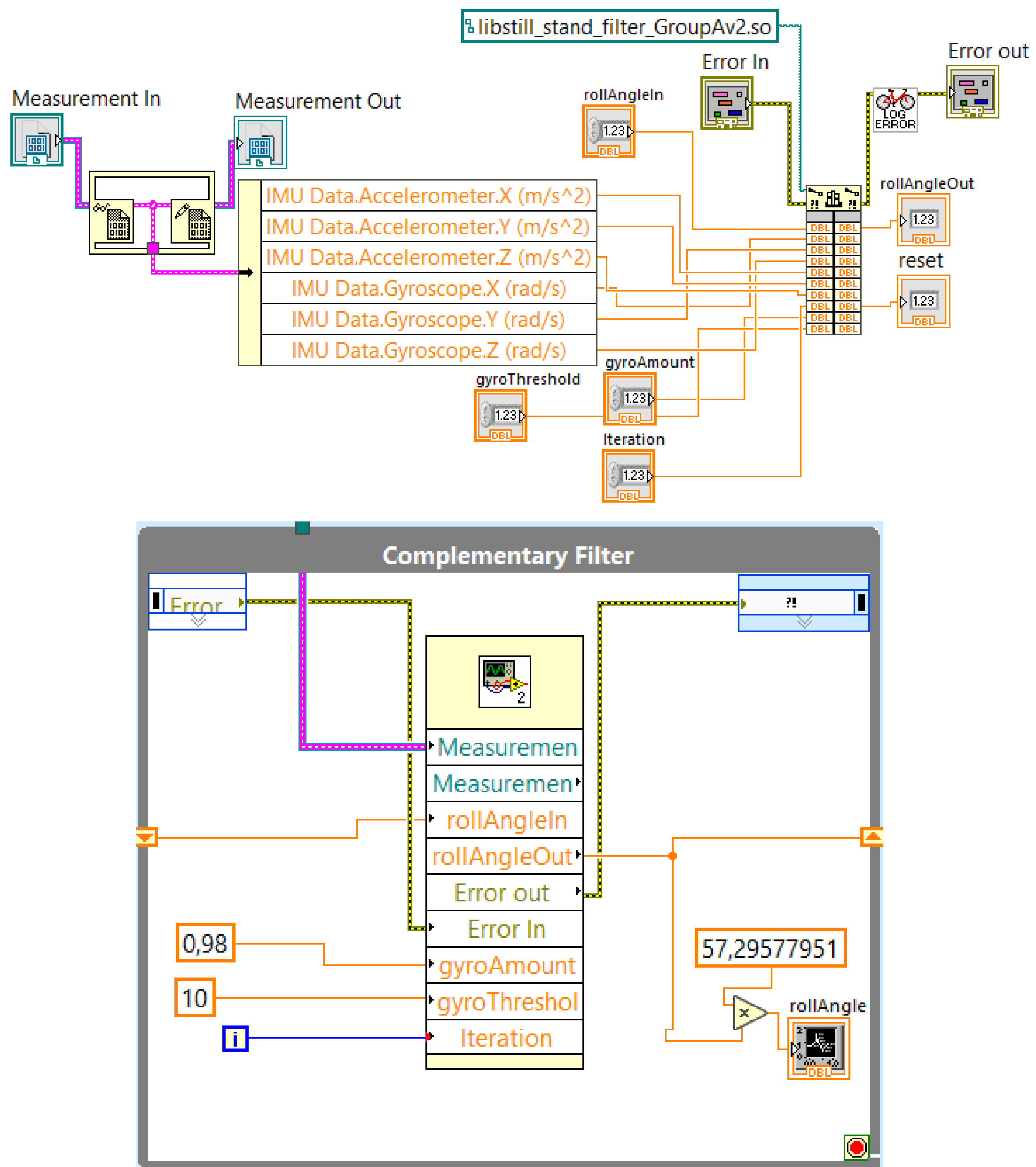


Figure 4.2: LabVIEW main- and sub-VI of the implementation of the complementary filter

4.2 Control loop

The control loop for the system controls the amount of current that the motor is given depending on the roll angle calculated by the complementary filter. Figure 4.3 describes the loop in which a cascade loop method is used. The inner control loop is handled internally by the VESC unit and the outer loop which supplies the VESC with a target current.

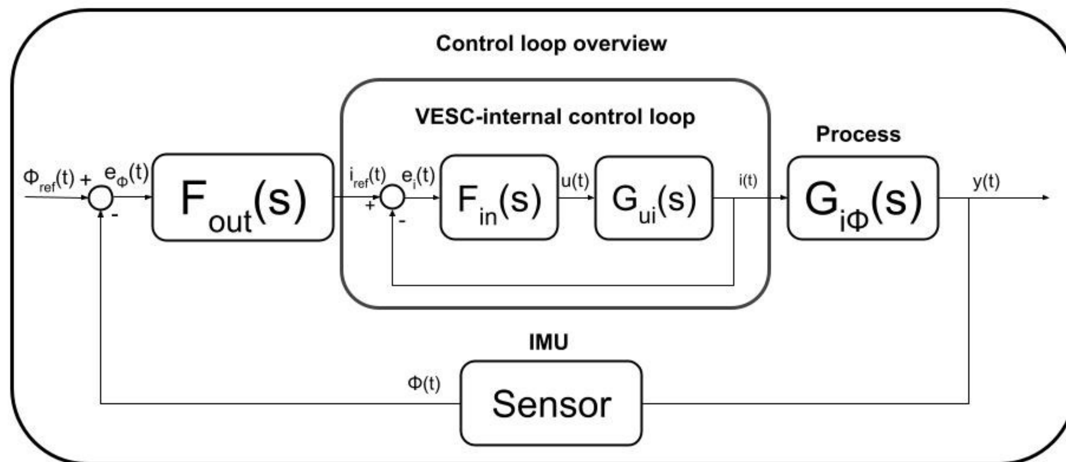


Figure 4.3: Cascade control loop of the system

4.2.1 Current control in LabVIEW

The first step in the control loop to balance the bicycle is calculating the amount of current needed by the motor, depending on the roll angle. This is managed in LabVIEW by a CLF-node with the real-time changeable variable of the roll angle. The C-code used in the CLF-node is referred to in Appendix A.4 and the functionality shown in Figure 4.4 describe how the PID values are set and how the target current is calculated. There is also a functionality implemented where the system dynamically adjusts the reference angle based on the angular velocity to prevent the system from accelerating to unreasonable speeds.

In the C-code, Appendix A.4, there is also implemented functionality to limit the amount of current that the control loop will give to the VESC so that the hardware does not break down. There is also a functionality to prevent windup of the integral part.

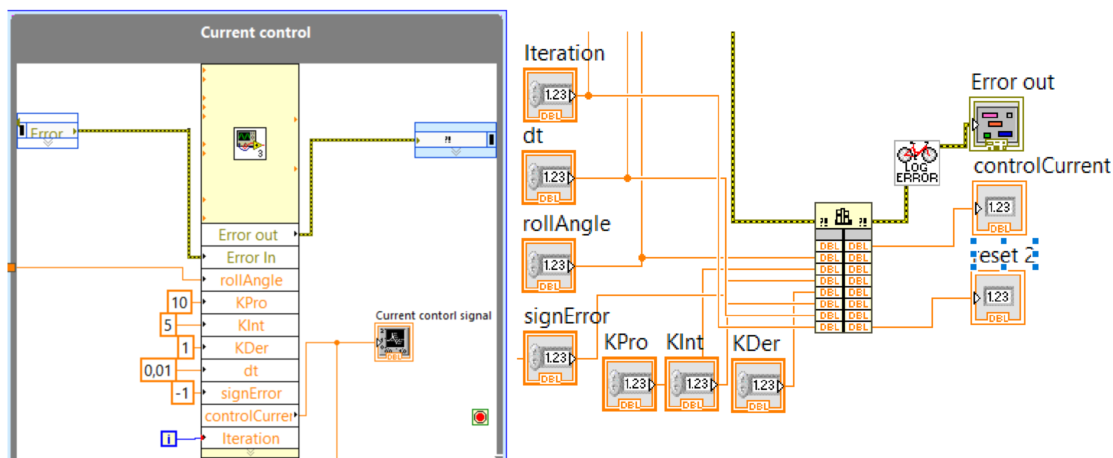


Figure 4.4: LabVIEW main- and sub-VI of the current control loop

4.2.2 Motor control in LabVIEW

The motor control in LabView is based on earlier solutions where the solution of driving the implemented forward motor from a VESC-unit, but reconnecting the phase cables from the unit to the balancing motor used in this project. This solution uses TX/RX (Transmit/Receive) protocol which sends the data of desired current through one cable to the VESC and internally, the VESC uses it's own control loop to make sure that the current matches what is calculated in the current control step in the outer control loop.

Figure 4.5: LabVIEW main VI motor control function

4.3 Full SIMULINK simulation with limitations

To evaluate the behavior of the complete system under realistic physical constraints, a full-scale Simulink model was developed. The purpose of this simulation environment was to test the stability of the controller, analyze performance limits and verify that the system remains robust when subjected to sensor noise, saturation's on motor current and voltage and discrete time sampling effects.

The model is divided into two main subsystems, the nonlinear process model and the angle-estimation block. The simulation uses the same sampling time as the physical IMU to accurately capture discretization effects, and the solver was set to Runge-Kutta 4 because of it's numerical stability and constant sampling time.

A cascade control structure was implemented to emulate the internal current regulation of the VESC motor controller. Since the VESC regulates current at a significantly lower bandwidth than the outer angle-correction loop, this structure provides a realistic representation of the real hardware behavior.

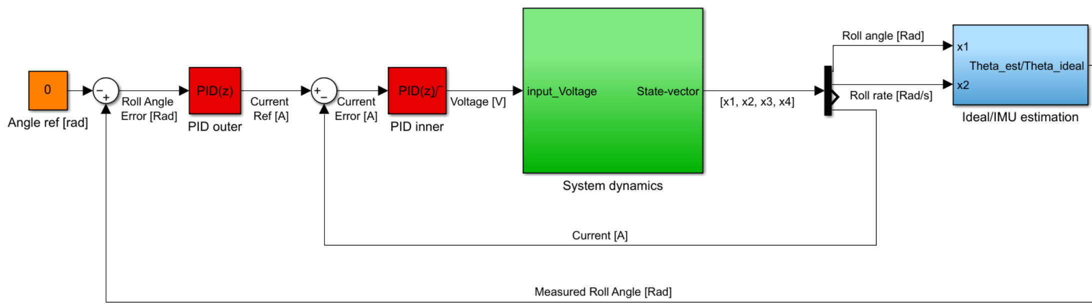


Figure 4.6: Non-linear simulation model in Simulink

4.3.1 Process - Subsystem

The process block was used to simulate the behavior of the system in the most accurate way possible. It uses voltage as input and saturates it according to hardware limitations. The saturated voltage is then fed into a function block that uses the nonlinear system derived in chapter 3.3, see Appendix A.1 . It outputs the derivative of the state vector which is then integrated to get the actual values. This vector represents the current states of the system and serves a dual purpose. Firstly, it is fed back into the function block to compute the state derivatives for the next time step. Secondly, it is the output of the process block, providing the necessary data for the rest of the system.

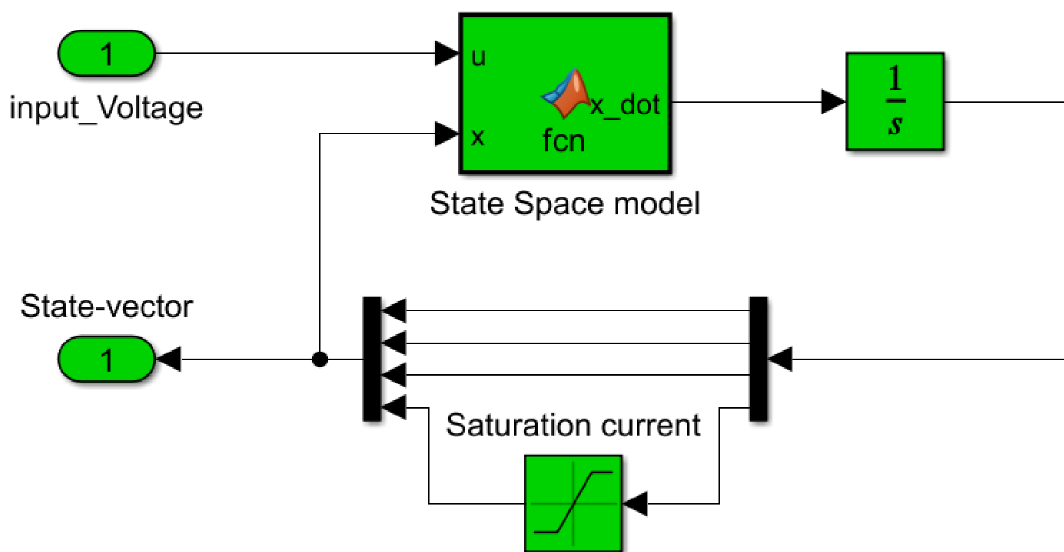


Figure 4.7: Process subsystem, simulated model of the bicycle

4.3.2 Angle estimation - Subsystem

The blue subsystem represents the estimation of the roll angle by simulating how an IMU works instead of using ideal measurements. The estimation is based on simulated roll angle and roll rate signals, which are processed using the same complementary filter algorithm as in the real system in Appendix A.3.

Noise is added to both the acceleration and angular rate signals. Appendix A.2 which is the code in the Simulink code block. Overall, this subsystem provides a realistic estimation of the roll angle, including noise and small errors similar to those of a real prototype.

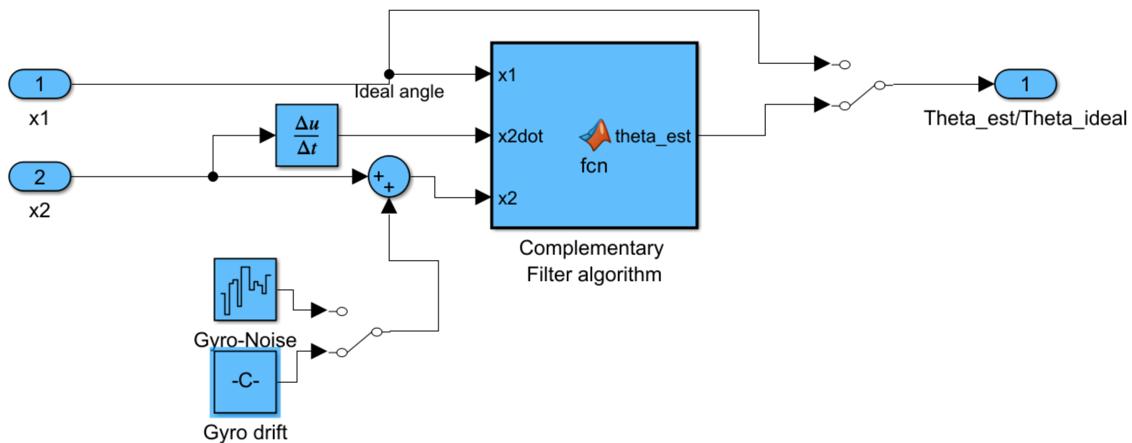


Figure 4.8: Angle estimation subsystem, simulated model of the complementary filter

4.3.3 Tuning

Since the cascade control structure consists of an inner current loop and an outer roll-angle loop, the complete controller contains up to six tunable parameters: $(K_p, K_i, K_d)_{inner}$ and $(K_p, K_i, K_d)_{outer}$. A Ziegler Nichols inspired tuning procedure was used to determine suitable parameter values for both loops.

The inner loop was tuned first, with the outer loop held at a constant value to isolate the current dynamics. The proportional gain K_p was gradually increased until the system reached the critical gain $K_{p,crit}$, defined as the point where sustained oscillations occur. Based on this value, an integral term K_i was introduced and increased until the steady-state error was eliminated. No derivative action was required for the current, as the PI structure provided sufficient stability and response time.

After tuning the inner loop, the outer roll-angle loop was connected to the full structure and the same procedure was applied. Unlike the inner loop, the outer loop required a derivative term K_d to achieve sufficient damping and to reduce sensitivity to measurement noise. The derivative action also allowed the settling time

to be adjusted without excessively increasing the proportional gain. A too aggressive response would have demanded more current through the motor than what was physically possible.

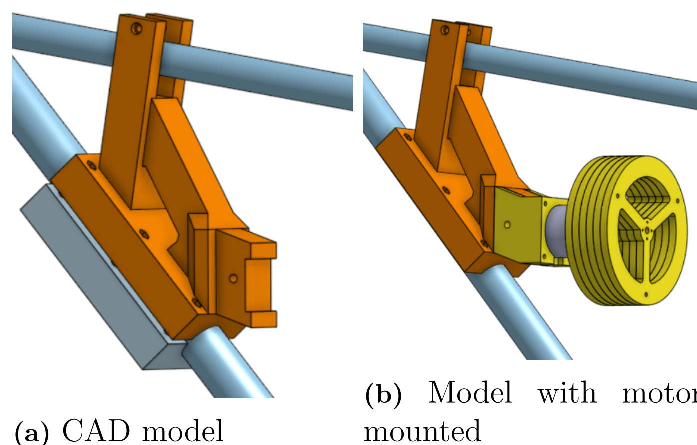
4.4 Standalone balancing control with an ESP32

As an alternative to the MyRIO-based implementation, the balancing controller was also implemented on the ESP32, using the Arduino IDE as described in Section 2.3.3. The roll angle is estimated using the same complementary filter described in Section 4.1, where the gyroscope and accelerometer data from the IMU are fused according to equation 4.3.

The output of the PID controller implemented is a current setpoint which is sent to the VESC over a serial communication interface using the VescUart library. The VESC then regulates the motor current internally. Anti-windup is implemented by removing the integral contribution whenever the output is saturated. A safety limit is also implemented such that the motor is stopped if the roll angle exceeds a defined threshold, beyond which the system can no longer recover. The implementation and which pins the IMU and VESC is connected to can be found in the code of the ESP32 in Appendix A.5.

4.5 Mount for motor on the black bike

When this project started the orange bike was the only one capable to have the balance motor mounted onto it. This was fixed by designing a mount for black bike with the CAD software Onshape. This was manufactured with a 3D-printer and mounted to the black bike. The shape of the mount is adapted to the geometry of the black bike and the equipment present. It was made with 10% infill as a quick way to solve to the issue.



(a) CAD model

(b) Model with motor mounted

Figure 4.9: CAD model of bike-motor attachment

5

Results

This section outlines the main results obtained from the conducted experiments. It highlights the measured performance, observed system behavior, and any irregularities identified during testing. The presented findings provide a structured overview that serves as a basis for further analysis and discussion.

5.1 Simulation Results

Three different error scenarios were tested independently using the full Simulink model to validate the system's performance. The results are presented in two plots, the first comparing the actual angle against the estimated angle, while the second displays the corresponding motor current and voltage. They all share the same PID parameters of $K_{p,outer} = 3000$, $K_{i,outer} = 500$, $K_{d,outer} = 400$, $K_{p,inner} = 0.2$, $K_{i,inner} = 4$, $K_{d,inner} = 0$ and have the same saturation limits of $70A$ and $22V$. They also illustrate the maximum possible initial offset angle for that specific scenario.

5.1.1 Time Varying Errors

The stationary reading from the IMU was tested, giving clear indications of what errors came from the accelerometer and gyro signals. The accelerometer showed errors with a standard deviation of approximately 0.0067 m/s^2 and the gyro showed errors with a standard deviation of approximately 0.0033 rad/s^2. These errors where implemented to simulate time varying errors and Figure 5.1 and Figure 5.2 shows the results.

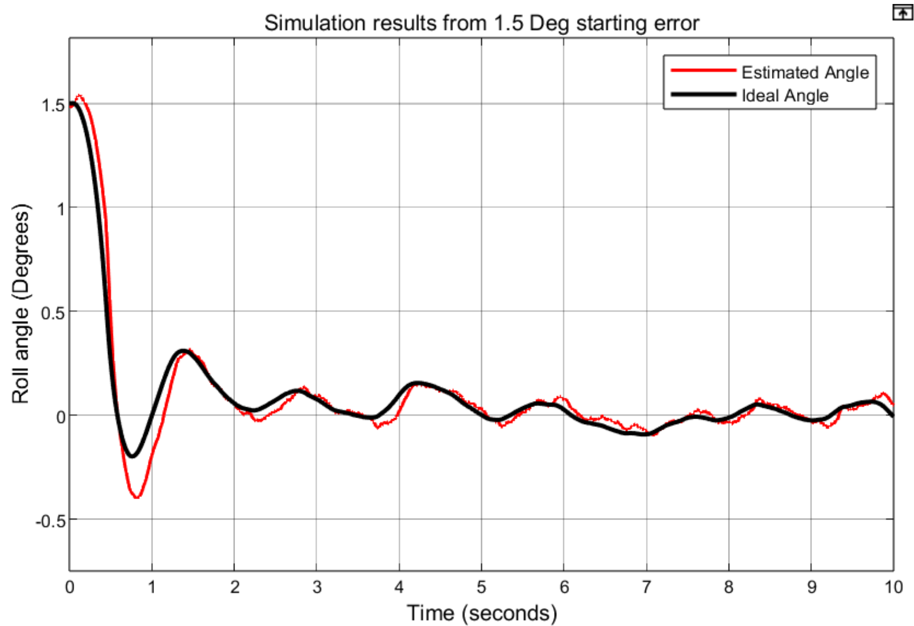


Figure 5.1: Simulink simulation showing how the roll angle changes with an initial 1.5° offset under time-varying accelerometer and gyroscope noise.

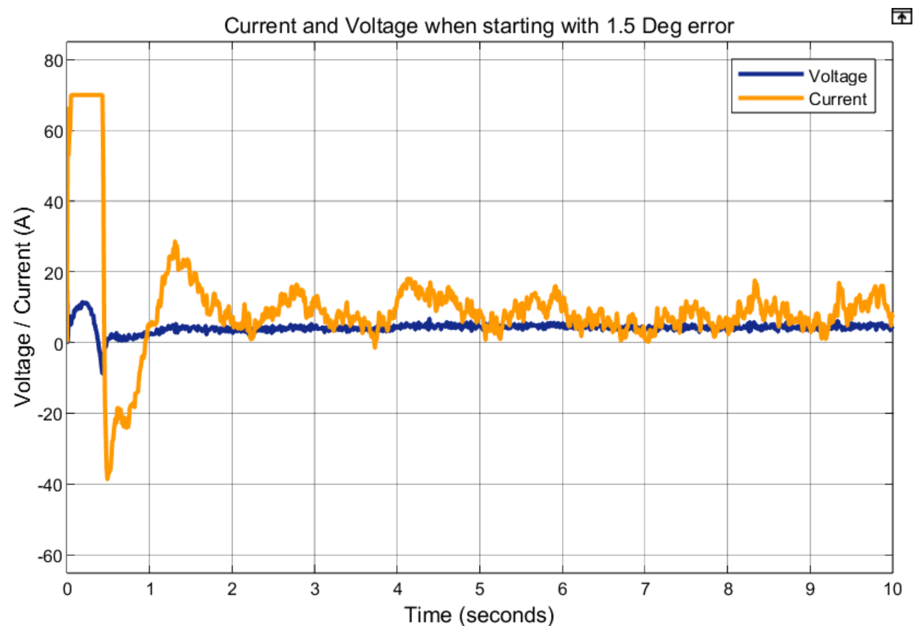


Figure 5.2: Simulink simulation showing how the current and voltage signals changes with an initial 1.5° offset under time-varying accelerometer and gyroscope noise.

5.1.2 Initial Offset Error

The stationary reading from the IMU was also tested to determine the effect of a constant measurement offset in the angle estimation, emulating a scenario where the bike does not receive an accurate initial angle reading. This simulation highlights

the importance of the accelerometer, as shown in Figure 5.3, where the estimated angle and the actual angle initially differ by 0.2°. The plot shows how the two curves converge to the same value over time, demonstrating that the controller is able to compensate for the initial IMU offset. Figure 5.4 shows the corresponding current and voltage over time for the same test. The controller was able to correct a maximum Roll angle offset of 1.46° when using an IMU measurement error of 0.2°.

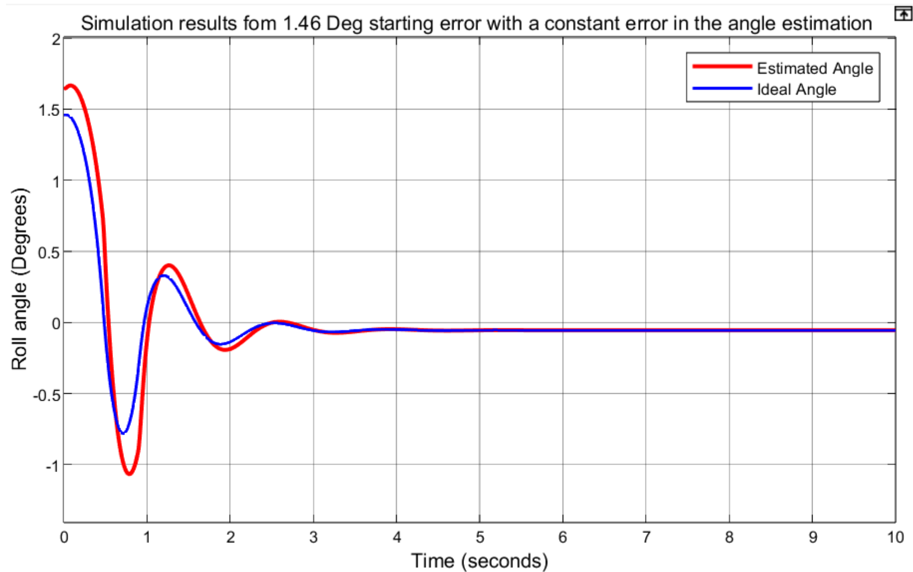


Figure 5.3: Simulink simulation showing how the roll angle changes with a 1.46° initial offset when the IMU starts with a constant measurement error of 0.2°.

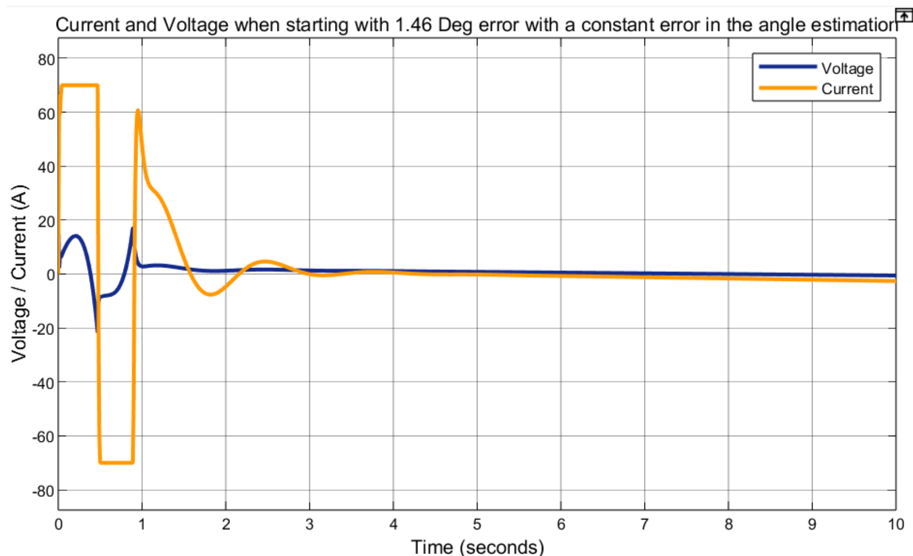


Figure 5.4: Simulink simulation showing how the current and voltage signals changes with a 1.46° initial offset when the IMU starts with a constant measurement error of 0.2°.

5.1.3 Gyro bias

A constant gyro bias was introduced to evaluate how the controller performs under steady state drift in the roll angle rate measurement. Figure 5.5 shows the roll angle response when the gyro is subjected to a $0.2°/s$ constant error, leading to a drift. The controller was able to correct an initial roll angle offset of up to $1.51°$. Figure 5.6 shows the corresponding current and voltage signal for the same test.

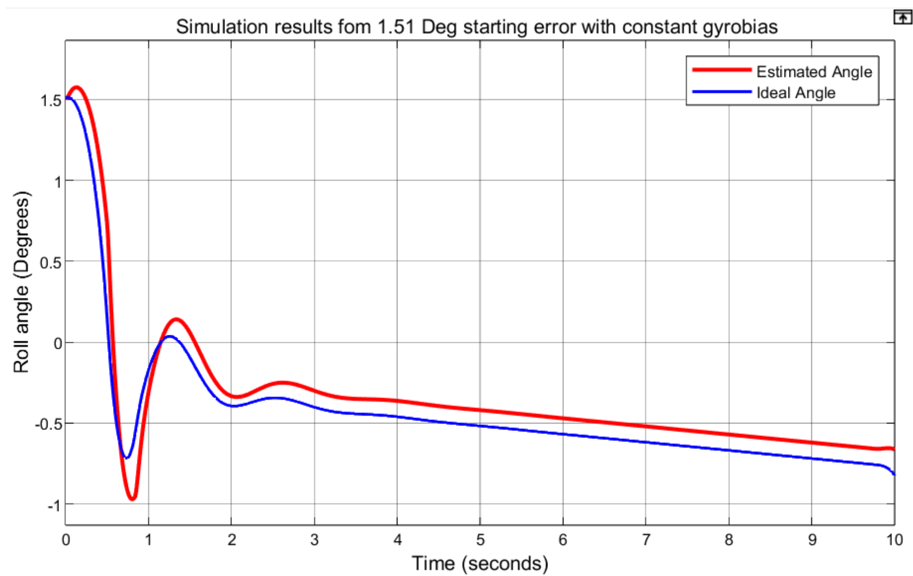


Figure 5.5: Simulink simulation showing how the roll angle changes with a $1.51°$ initial offset when the IMU is affected by a constant gyro bias of $0.2°/s$.

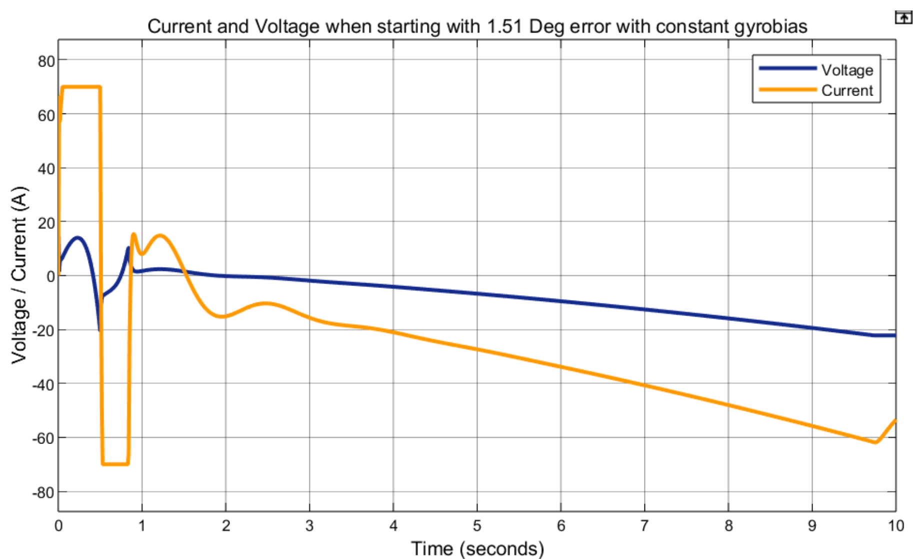


Figure 5.6: Simulink simulation showing how the current and voltage signals changes with a $1.51°$ initial offset when the IMU is affected by a constant gyro bias of $0.2°/s$.

5.2 Performance of Complementary filter

The need for a complementary filter was tested by comparing different values of the constant α. This was done by controlling the ideal roll angle while plotting both the ideal and estimated roll angle. $\alpha = 0$ shows the case where only the accelerometer is used, causing high frequency error, while maintaining correct mean value. $\alpha = 1$ shows the case where only the gyro is used, causing drift over time. The alpha that gave the best combination of short term precision and low drift was $\alpha = 0.98$, this is shown in Figure 5.7:

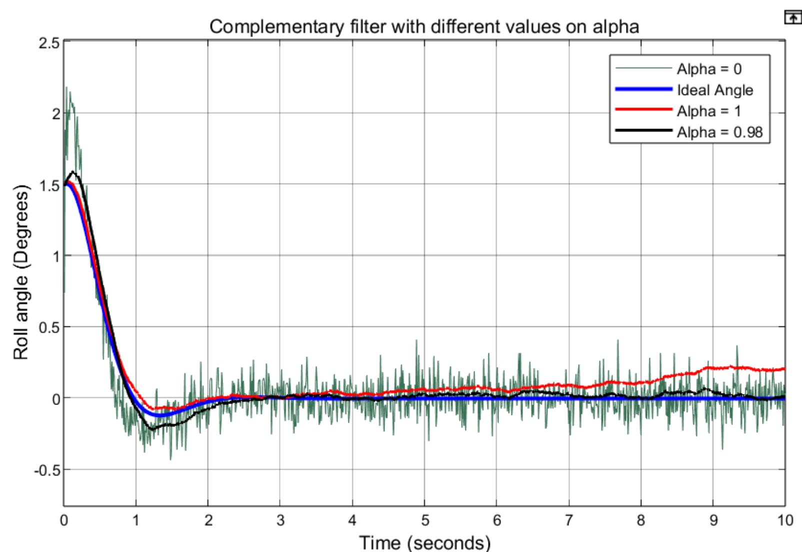


Figure 5.7: Complementary filter with different values on α

5.3 Balancing Results

The PID parameters obtained from Simulink simulations gave poor results. Simulation results were therefore not of much use and a new systematic PID tuning had to be done. During testing, speeds higher than 667 rpm caused high vibrations and was deemed a safety risk. Therefore, a max rpm was set which caused the motor to saturate within less than 2 seconds of full acceleration.

A proportional control factor between P = 100 to P = 400 managed to make the bike stand longer than natural, and in the best cases making the bike balance for up to 5 seconds. Proportional factors of 140 to 180 caused self induced oscillation. One observation was that values of 200 and above caused mostly saturated control signals. When also using an integral factor of 10-100 there seemed to be some signs of the gyro slowing down.

A derivative factor of 10-200 dampened oscillations in some sense, but also introduced instability. A low pass filter was implemented with some effect but did not manage to solve the issue.

6
Conclusion

This chapter will summarize the findings during the project. The project has been heavily affected by earlier development and problems that could not be predicted. This means that most of the problems that are solved are theoretical and only a few solutions have been implemented and tested.

6.1 Discussion

Much of this project was spent trying to understand the existing structure of the LabVIEW code. This in combination with not being sure which bicycle we were allowed to work on lead a lot of time spent on systems that were not used in the final product. When the bike on which we were allowed to work was clear, the problem of software compatibility arose. This lead to the decision to change the solution to the ESP32 in our system which solved problems that we were not able to solve with LabVIEW.

6.2 Simulation vs. Real System

The significant difference in the tuning parameters used during testing can be attributed to multiple factors. One is the delay between the physical change of roll angle and the motor response. This is the sum of all stages in the control process each contributing with delay. The delay makes the bike tilt more before the motor responds. This requires a stronger response compared to a simulation where there are no delays.

Another source of error not considered in the simulations were vibrations. We assume this was primarily caused by the gyro being slightly off-center, and that this issue was further amplified by the attachment being flexible, allowing further resonance. These could contribute estimated angle errors of up to +-0.3 degrees for every turn of the gyro. When the derivative factor was introduced in the tests the bike became less stable contrary to the simulations. A low pass filter was introduced as a final effort to fix this which gave vague signs of improvement, but not success. This gave the unexpected and unsatisfying result of achieving the longest (but not necessarily the most robust) stability without the derivative control factor.

It was known early in the project that the system would be weak and only capable of handling small angles. While this was reflected in the simulations, the cascade

of issues deriving from limits and accuracy was not. The amount of proportional factor necessary for self induced oscillation and stabilization was remarkably higher than estimated in the simulations. This meant that the output signal was saturated most of the time, even when the changes in angle were small.

A more precise estimation of the angle would likely give the possibility of stability within the narrow angle limits. Additional precision would be needed in order to calm the control signal enough so that the motor avoids overheating during long runs. As during our testing, breaks had to be taken to cool the motor.

The alpha factor was changed in the ESP32 implementation. This was because the sampling rate was increased to 500 Hz in order to improve the reaction time. This made the initial value of alpha too low, and it had to be increased in order to make the estimation more stable. There was also an issue with the angle shifting in the opposite direction during induced acceleration of the bike. Higher values of alpha fixed this, but may have increased the risk of drift. One interesting consideration is that the vibration issue mentioned may have confused the gyroscope readings and made the issue even worse and this by itself could be a reason to make controllability impossible within the narrow limits.

6.3 Further development

There are several potential developments that can be done and a few potential improvements have been identified and are formulated below. One possible development is to resolve the issues with LabVIEW and implement the ESP32 solution on the myRIO with RX/TX communication to the VESC.

Another development that is possible is to continue on the ESP32 or similar embedded system and implement all of the other capabilities onto it. This will have one advantage which is that the system could become more compact.

A third development needed for the system to be able to balance itself in a wider range of motion will be to change the motor and gyro. This could be done with having a motor that is stronger and can give more torque and a gyro with larger moment of inertia. It will allow the gyro to experience the same amount of torque for longer without saturating.

The mount made for the bicycle was weak and it would be wise to make a new one out of a stronger material like PETG instead of PLA and with higher infill, before more testing is performed. This would most likely decrease vibrations as well.

Another improvement could be to connect two batteries in parallel to primarily increase the recharge current. This could double the braking torque. These increased currents could be used in short bursts but a temperature control and estimation system would be wise to implement in order to prevent overheating.

Bibliography

[1] A. Rönnmark, P. Fazekas, M. D. A. Hussein, and S. Giragos, "Autobike: Autonomous balancing project," Mälardalens University, Västerås, SE, 2023.

[2] E. Andersson and H. Hultergård, "Programming a self driving bike," Chalmers University of Technology, Gothenburg, SE, 2022. Accessed: Apr. 6, 2026. [Online]. Available: `https://odr.chalmers.se/server/api/core/bitstreams/3a05c0f1-8c5a-44b2-be15-7ff87947fe6f/content`.

[3] M. Grieco, "Experimental validation of an autonomous bike control system," Università degli Studi del Sannio, Benevento, IT, 2023.

[4] National Intruments. "Myrio-1900 getting started guide and specifications," Accessed: Apr. 16, 2026. [Online]. Available: `https://www.ni.com/docs/en-US/bundle/myrio-1900-getting-started/resource/376047d.pdf?srsltid=AfmBOopKk3KW7hTJYU92-xkyAeXK3PqGzc36MpPT97D4XTqq6yIR8VZW`.

[5] N. Kayiranga, "Internship report," École Polytechnique de l'Université de Nantes, Nantes, FR, 2021.

[6] Model Motors. "Axi 5360/24hd v3 gold line specifications," Accessed: Apr. 16, 2026. [Online]. Available: `https://www.modelmotors.cz/product/detail/508/`.

[7] Spintend. "What's vesc? how to use it ? what's difference between vesc and esc?" Accessed: Apr. 16, 2026. [Online]. Available: `https://spintend.com/blogs/news/whats-vesc-how-to-use-it-whats-difference-between-vesc-and-normal-esc`.

[8] Oskar Fällman. "What is vesc (complete guide)," Accessed: Apr. 16, 2026. [Online]. Available: `https://fallman.tech/vesc/`.

[9] Digilent. "Pmod nav (legacy)," Accessed: Apr. 16, 2026. [Online]. Available: `https://digilent.com/reference/pmod/pmodnav/start?redirect=1`.

[10] "Battery." [Preprint], Accessed: Apr. 16, 2026. [Online]. Available: `https://hobbyking.com/en_us/turnigy-high-capacity-14000mah-6s-12c-multi-rotor-lipo-pack-w-xt90.html`.

[11] I. Poole. "What is labview?" Accessed: Apr. 6, 2026. [Online]. Available: `https://www.electronics-notes.com/articles/test-methods/labview/what-is-labview.php`.

[12] I. Poole. "Labview vi: Virtual instrument," Accessed: Apr. 6, 2026. [Online]. Available: `https://www.electronics-notes.com/articles/test-methods/labview/vis-virtual-instruments.php`.

[13] B. O'Neill. "What is a shared object file," Accessed: Apr. 6, 2026. [Online]. Available: https://cloudmersive.com/article/What-is-a-Shared-Object-File.

[14] V. Hunter Adams. "Complementary filters," Accessed: Mar. 28, 2026. [Online]. Available: https://vanhunteradams.com/Pico/ReactionWheel/Complementary_Filters.html.

[15] Digilent. "Pmod nav reference manual," Accessed: Mar. 28, 2026. [Online]. Available: https://digilent.com/reference/_media/reference/pmod/pmodnav/pmod_nav_rm.pdf.

A

Code

In this appendix, the code used in simulations and implementations is written and simplified as much as possible.

A.1 Simulations

Listing A.1: MatLab code for the state space model used in the green subsystem in Simulink

```
function x_dot = fcn(u, x)

m_g = 7;
r = 0.15;
m_t = 40;
g = 9.82;
h = 0.5;
J_t = m_t * h^2;
J_g = 0.5 * m_g * r^2;
b = 0.02;
K_m = 0.1;
K_u = 0.1;
L = 0.003;
R_m = 0.068;

x1 = x(1);
x2 = x(2);
x3 = x(3);
x4 = x(4);

dx1 = x2;
dx2 = (1/J_t) * ( m_t*g*h*sin(x1) - K_m*x4 + b*x3 );
dx3 = (1/J_g) * ( K_m*x4 - b*x3 );
dx4 = (1/L) * ( u - R_m*x4 - K_u*x3 );

x_dot = [dx1; dx2; dx3; dx4];
end
```

Listing A.2: Matlab code for the IMU estimation algorithm used in the blue subsystem in Simulink

```
function theta_est = fcn(x1, x2dot, x2)

persistent alpha dt g l theta_prev theta_bias

if isempty(alpha)
    alpha = 0.98;
    dt = 0.01;
    g = 9.82;
    l = 0.5;
    theta_bias = 0;

    theta_prev = x1 + theta_bias;
end

fy_body = g*cos(x1) - x2^2*l;
fx_body = g*sin(x1) - x2dot*l;

gyro = x2;
ax_noisy = fx_body + 0.02*randn;
ay_noisy = fy_body + 0.02*randn;

theta_acc = atan2(ax_noisy, ay_noisy);

theta_est = alpha*(theta_prev + gyro*dt) + (1-alpha)*
   theta_acc;

theta_prev = theta_est;
```

A.2 Roll angle calculation, complementary filter

Listing A.3: C code for the roll angle calculation

```
#include <math.h>

double dt = 0.01;
static int initCounter = 0;
static double accelSum = 0.0;

extern void comp_filter(double *rollAngle, double imuRateX,
    double accelY, double accelZ, double reset, double
    gyroAmount){
    if (reset == 0) {
        initCounter = 0;
        accelSum = 0.0;
    }
    double accelRoll = atan2(accelY, accelZ); // Calculate
        roll angle from acceleration

    // Initialize roll angle
    if (initCounter < 100){
        accelSum += accelRoll;
        initCounter++;
        if (initCounter == 100){
            *rollAngle = accelSum / 100.0;
        } return;
    }
    // Complementary filter
    *rollAngle = gyroAmount * (*rollAngle + imuRateX * dt) +
        (1.0 - gyroAmount) * accelRoll;
}
```

A.3 Control signal calculation

Listing A.4: C code for the calculation of the control signal from roll angle

```c
#include <math.h>

// Roll control states
static double integral = 0.0, prevErr = 0.0;

extern void angle_control_signal(double *targetCurrent,
    double rollAngle, double omega, double Kp, double Ki,
    double Kd, double signError, double omegaPhiRef, double dt
    , double reset){
    if (reset == 0.0){
        integral = 0.0, prevErr = 0.0;
    }
    // Roll control loop
    double setpointRoll = omegaPhiRef * omega;
    double errorRoll = signError *(setpointRoll - rollAngle);
    integral += errorRoll * dt;
    double derivate = (errorRoll - prevErr) / dt;
    double targetCurrentValue = (Kp * errorRoll) + (Ki *
        integral) + (Kd * derivative);
    if (targetCurrentValue > maxAmp){
        *targetCurrent = maxAmp;
        integral -= errorRoll * dt; // Reset integral term to
            prevent windup
    } else if (targetCurrentValue < -maxAmp){
        *targetCurrent = -maxAmp;
        integral -= errorRoll * dt;
    } else{
        *targetCurrent = targetCurrentValue;
    }
    // If the roll angle exceeds the maximum allowed angle,
        set target current to 0 to prevent further movement
        and potential damage
    if (rollAngle > maxAngle || rollAngle < -maxAngle){
        *targetCurrent = 0.0;
    }
    // Update state
    prevErr = errorRoll;
}
```

A.4 Implementation in ESP32

Listing A.5: C++ code for the implementation to the ESP32 hardware

```
#include <Wire.h>
#include <SparkFunLSM9DS1.h>
#include <VescUart.h>

LSM9DS1 imu;
VescUart UART;

// IMU
double alpha = 0.999;
double beta = 0.05;

// PID
double K_p = 160;
double K_i = 0;
double K_d = 5;

double angleOffset = 0;

// Plottable values
double angle = 0.0;
double setpointRoll = 0;
double errorRoll = 0;
double integral = 0;
double omega = 0;
double omegaPhi_int = 0;
double targetCurrent = 0;
double filteredDerivative = 0;
bool vesc_retr = false;

int count = 0;
int sampleN = 0;
double prevErr = 0;

double maxAmp = 60;
double maxAngle = 10;

// Experimental omega to phi_ref factor (unused)
double omegaPhi_P = 0;
double omegaPhi_I = 0;

void setup() {
  Serial.begin(921600);

  // // --- IMU ---
  Wire.begin(21, 22);
```

```
46    if (!imu.begin()) {
47      Serial.println("IMU kunde inte starta");
48      while (1);
49    }
50    Serial.println("IMU OK");
51
52   // --- VESC ---
53    Serial1.begin(115200, SERIAL_8N1, 25, 26);
54    while (!Serial1) { ; }
55    /** Define which ports to use as UART */
56    UART.setSerialPort(&Serial1);
57    Serial.println("VESC OK");
58  }
59
60
61  // ----- Functions for live debbuging and configuration -----
62  // Commands are: <var>=<value> and get <var>
63  const int varN = 9;
64  double *variableList[varN] = { &K_p, &K_i, &K_d, &omegaPhi_P,
        &omegaPhi_I, &alpha, &angleOffset, &maxAmp, &beta };
65  String variableNames[varN] = { "K_p", "K_i", "K_d", "
       omegaPhi_P", "omegaPhi_I", "alpha", "angleOffset", "maxAmp
       ", "beta" };
66  String variableCalls[varN] = { "p", "i", "d", "op", "oi", "
       alpha", "aoffs", "maxa", "beta" };
67
68  // Commands are: plot on <plotvar> and plot off <plotvar>
69  const int plotN = 10;
70  double *plotList[plotN] = { &angle, &setpointRoll, &errorRoll
       , &integral , &omega, &omegaPhi_int, &targetCurrent, &
       filteredDerivative, (double*)0, (double*)0 };
71  String plotNames[plotN] = { "angle", "setpointRoll", "
       errorRoll", "integral", "omega", "omegaPhi_int", "
       targetCurrent", "filteredDerivative", "count", "vesc_retr"
        };
72  bool plotEnable[plotN] = { false, false, false, false, false,
        false, false, false, false, false };
73
74  void setVariable(String call, double val) {
75      call.toLowerCase();
76      for (int i = 0; i < varN; i++) {
77          if (variableCalls[i] == call) {
78              *(variableList[i]) = val;
79              Serial.println("Changed " + variableNames[i] + "
                   to " + String(val));
80              return;
81          }
82      }
83      Serial.println("Not found");
```

```
84  }
85  void getVariable(String call) {
86      call.toLowerCase();
87      for (int i = 0; i < varN; i++) {
88          if (variableCalls[i] == call) {
89              Serial.println(variableNames[i] + " is " + String
                   (*(variableList[i])));
90              return;
91          }
92      }
93      Serial.println("Not found");
94  }
95  void setPlot(String name, bool val) {
96    for (int i = 0; i < plotN; i++) {
97          if (plotNames[i] == name) {
98              plotEnable[i] = val;
99              Serial.println("Set plot for " + plotNames[i] + "
                    to " + String(val));
100             return;
101         }
102     }
103     Serial.println("Not found");
104 }
105 void plotAll(){
106   bool hasPlotted = false;
107   for (int i = 0; i < plotN; i++) {
108         if (plotEnable[i]) {
109             if (plotNames[i] == "count") Serial.print(
                   plotNames[i] + ":" + String(count) + ",");
110             else if (plotNames[i] == "vesc_retr") Serial.
                   print(plotNames[i] + ":" + String(vesc_retr) +
                    ",");
111             else Serial.print(plotNames[i] + ":" + String(*(
                   plotList[i])) + ",");
112             hasPlotted = true;
113         }
114     }
115     if (hasPlotted) Serial.println();
116 }
117 void readSerialInput() {
118     String str = Serial.readString();
119     str.trim();
120     int index = str.indexOf('=');
121     if (index > 0) {
122         String msg = str.substring(0, index);
123         String valStr = str.substring(index+1);
124         double val = valStr.toDouble();
125         setVariable(msg, val);
126     } else {
```

```
      if (str.length() > 4 && str.substring(0,3) == "get") {
        getVariable(str.substring(4));}
      if (str.length() > 8 && str.substring(0,7) == "plot on"
         ) {
        setPlot(str.substring(8), true);}
      if (str.length() > 9 && str.substring(0,8) == "plot off
         ") {
        setPlot(str.substring(9), false);}
    }
}
// --------------------------------------------------------

double t = 0;
void loop() {
  double dt = 1.0 / 500.0;

  // --- IMU ---
  imu.readGyro();
  imu.readAccel();

  double gx = imu.calcGyro(imu.gx);
  double ay = -imu.calcAccel(imu.ay);
  double az = imu.calcAccel(imu.az);

  double accAngle = degrees(atan2(ay, az));
  double gyroAngle = angle + gx * dt;

  angle = alpha * gyroAngle + (1.0 - alpha) * accAngle;

  // Integrate experimental omega to phi_ref factor (unused)
  omegaPhi_int += omega * omegaPhi_I * dt;
  if (abs(omegaPhi_int) > 2) {
    omegaPhi_int = omegaPhi_int / abs(omegaPhi_int) * 2;
  }

  // Our control loop
  double signError = 1.0;
  setpointRoll = omega * omegaPhi_P + omegaPhi_int;
  errorRoll = signError * (setpointRoll - angle + angleOffset
     );

  integral += errorRoll * dt;
  double derivative = (errorRoll - prevErr) / dt;

  filteredDerivative = (beta * derivative) + (1 - beta) *
     filteredDerivative;

  targetCurrent =
```

```
171      (K_p * errorRoll) + (K_i * integral) + (K_d *
            filteredDerivative);
172
173    // Saturation + a n t i windup
174    if (targetCurrent > maxAmp) {
175      targetCurrent = maxAmp;
176      integral -= errorRoll * dt;
177    } else if (targetCurrent < -maxAmp) {
178      targetCurrent = -maxAmp;
179      integral -= errorRoll * dt;
180    }
181    prevErr = errorRoll;
182
183    // Safety cutoff
184    if (angle > maxAngle || angle < -maxAngle) {
185        targetCurrent = 0.0;
186    }
187
188    // Average angle in the beggining
189    if (t > 1.0 && t < 6.0) {
190      angleOffset += angle;
191      sampleN++;
192    } else if (sampleN > 0) {
193      angleOffset /= (double)sampleN;
194      sampleN = 0;
195    }
196
197    plotAll();
198
199    // Set motor currents
200    if (t > 8) {
201      if (count % 10 == 0) {
202        UART.setBrakeCurrent(targetCurrent);
203        UART.setCurrent(targetCurrent);
204
205        // retrieve and request are custom functions to read
              vesc data without causing delay
206        vesc_retr = UART.retrieveVescValues();
207        omega = UART.data.rpm;
208        UART.requestVescValues();
209      }
210    }
211
212    // Read input from serial and process it
213    if (count % 100 == 0 && Serial.available()) {
          readSerialInput();}
214    count += 1;
215    t += dt;
216  }
```

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY