Report: Selfbalancing motorcycle

Part of module

Cyber physical systems

Name	Student number	Email address
Ân Lê	1647036	ph.le@student.han.nl
Anurag Shinde	2132290	ad.shinde@student.han.nl
Raymond Woudenberg	1548568	r.woudenberg@student.han.nl

Deadline project: 13 June 2024
Company: HAN University of Applied Sciences
HAN supervisor: Hatim Mala

Contents

PREFACE

This project plan has been written in the Cyber Physical Systems module of the Master Engineering System Programme.
During this project, we worked on a motor that must be self-balancing while driving.

We have experienced this project as fun and educational. We have come across many aspects of creating a control system for a motor.

Finally, we would like to thank Hatim Mala for his supervision during this project.

June 2024

Summary

The self-balancing motorcycle project aims to develop a control system that allows a motorcycle to maintain balance autonomously. The project is done by students of HAN University of Applied Science using the Arduino Engineering kit V2. The primary objectives include designing, testing, and optimizing the control system to achieve stable balance, straight motion, and steering.

In the initial phase, the control system was designed using pole placement and LQR matrix to achieve the desired stability and performance. Theoretical models suggested placing poles further to the left to minimize overshoot, although this required more control effort. However, real-world testing revealed that the motorcycle could not balance with the theoretical gains.

Through iterative testing and adjustment of the proportional (K_p), derivative (K_d), and inertia wheel (K_{pw}) gains, optimal gains were identified to achieve stable balance. This process involved multiple tests and fine-tuning to find the best balance position, resulting in successful self-balancing of the motorcycle.

Testing the motorcycle's straight motion presented additional challenges. To enable free movement, the setup required a wireless communication protocol, such as Wi-Fi. Although the Wi-Fi connection initially worked well after disconnecting the USB cable, it became unstable after 30 to 60 seconds. This instability highlighted the need for further improvements in the wireless communication setup to achieve reliable performance during straight-motion testing.

In conclusion, the project successfully developed a self-balancing motorcycle control system through iterative gain adjustment and real-world testing. The next steps involve stabilizing the Wi-Fi connection to allow for uninterrupted straight motion testing, ensuring the motorcycle can operate independently of the proximity constraints imposed by the USB connection.

Nomenclature

Symbols

Symbols	Description
l	length
F	force
m	mass
a	acceleration
r	radius
h	height
R	radius inertia wheel
τ	torque
θ	angle
$\ddot{\theta}$	angular acceleration
I	moment of inertia
ω	rotational speed
$\dot{\omega}$	rotational acceleration.
sin	sinus

Abbreviations

Abreviation	Description
N	Newton
m	meter
mm	millimeter
kg	kilogram
m/s^2	meter/$second^2$
rad/s^2	radians/$second^2$
Nm	Newtonmeter
kg/m^2	kilograms/$meter^2$
LQR	Linear quadratic regulator
CARE	Control Algebraic Riccati Equation
ZOH	Zero Order Hold

Chapter 1

Introduction

This section introduces the reader to the background of this project, defines the problems to be solved and mentions the project objectives and research questions to be explored.

1.1 Background

The Self-Balancing Motorcycle project consists of a two-wheeled robot that can balance and move using a rotating disc (inertia wheel) to compensate if the motorcycle loses balance. The motorcycle is controlled by an Arduino Nano 33 IoT, the Arduino Nano Motor Carrier, a DC motor to move the back wheel, DC motor with encoder to control the inertia wheel and a standard servo motor to steer the motorcycle's handle. [1]

A project with already functioning models exists on the Arduino Engineering kit website that provides the MATLAB Simulink files for the balancing of the motorcycle in one place. The website goes through the entire design flow to implement a controller to solve the problems of balancing the motorcycle. It explains stepwise the following:

1. Derivation of equations based on physical principles, primarily relying on Newton's laws.
2. Implementing equations in software to simulate the system behavior.
3. Developing a model using Simscape.
4. Designing a feedback controller and testing it in the loop with a Simscape model.
5. Implementing the feedback controller on real hardware.

1.2 Problem Definition

The motorcycle right now can only balance on one place using the inertia wheel. The feedback controller designed in Simulink only functions for balancing in one place. The motorcycle additionally should also be able to have the capability of straight motion and steering to make turns, all while balancing itself with the help of the inertia wheel.

1.3 Project Objective

There are two important project objectives and it is required to design non-linear control methods to achieve these objectives:

- Balancing with Straight Motions:

  Balancing with straight motion is the next logical extension of the feedback controller. Just like the inertia wheel, the rear wheel motor is driven by a DC motor and is equipped with an encoder. This motor needs to be integrated into the Simulink model.

- Balancing with Steering:

  While steering the motorcycle, to make turns we need to think about the extra torque component applied to the motorcycle about the wheel-ground axis. This extra "turning" torque may be comparable to the torque created by the inertia wheel. Hence, it will become important to modify the controller in such a way that we take advantage of this, so that the motorcycle leans inward as it turns.

1.4 Research question(s)

Main question:

- How can non-linear control strategies be effectively designed and implemented to ensure the stability of the self-balancing motorcycle during straight motion and turning maneuvers?

Sub questions:

- What are the impacts of sensor noise and data inaccuracies on the performance of the self-balancing motorcycle, and how can these impacts be reduced using filtering techniques?
- To what extent can the simulation model accurately predict the real-world behavior of the self-balancing motorcycle?

1.5 Approach

For successful modeling of the self-balancing motorcycle system, the appropriate state feedback of parameters should be determined. For this approach, the state feedback is determined by using different methods to find the optimal solution that meets both theory and practical. Then the self-balancing motorcycle is controlled to have straight motion and turning motion by Wifi. The subsequent content will explain these parts clearly.

1.6 Outline of minor project report

1. Introduction
2. Literature survey
3. Methodology
4. Results
5. Discussion
6. Conclusion

Chapter 2

Literature survey

2.1 Pole placement

Pole placement is a control method assigned to arbitrarily closed loop poles by state or output feedback [4]. Pole placement techniques are applicable to MIMO systems. This is a powerful technique in control system design used to place the closed-loop poles of a system at desired locations in the s-plane (Laplace domain) by designing an appropriate feedback controller. Closed-loop pole locations have a direct impact on time response characteristics such as rise time, settling time, and transient oscillations. By using pole placement techniques, design dynamic compensators can be designed.

2.2 Feedback linearization

Feedback linearization is an approach to nonlinear control design to algebraically transform a nonlinear system dynamics into a linear one [6]. This method involves designing a control input compensating for the nonlinear terms, resulting in a system that behaves linearly concerning the new input. By doing so, standard linear control techniques can be applied to achieve desired performance and stability.

Feedback linearization produces a linear model by the use of nonlinear coordinate transformations and nonlinear state feedback [5]. Subsequently, a nonlinear state feedback control law is implemented to achieve a linear relationship between the input and the output of the system. This approach enables the application of linear control techniques to nonlinear systems, facilitating the design of controllers that can achieve precise and robust performance. Feedback linearization is particularly useful in applications where precise control of nonlinear systems is required, such as robotics, aerospace, and chemical process control.

2.3 Linear Quadratic Regulator (LQR)

In the referenced literature an LQR controller was designed to determine appropriate values for the state feedback control gain matrix 'k' for the similar self-balancing motorcycle project using the electrodynamic equations of motion. LQR is the optimal theory of pole placement method as it defines the optimal pole location based on a quadratic cost function.

The quadratic cost function for the LQR controller in this paper was:

$$J(x, u(t)) = \frac{1}{2} \int_0^T \left(x^T Q x + r u(t)^2 \right) dt \tag{2.1}$$

where, $Q = Q^T$ and Q 0, $r > 0$ are the penalty parameters.

Control Algebraic Riccati Equation (CARE):

$$A^T P + PA - Pbr^{-1} b^T P + Q = 0 \tag{2.2}$$

Matrix Q and R:

Matrix Q expresses the penalty for the speed at which each of the parameters in the state vector should be reached. For instance, if matrix gives a bigger penalty for one state parameter and smaller for another, it punishes the first state parameter for reaching slower than the second. Matrix R specifies the penalty for using up control effort u(t). So, as bigger as the parameters of R gets, the more expensive the application of the control u(t) becomes.

The matrix Q was set to a 4x4 identity matrix and r = 100. The CARE was solved in MATLAB using the 'lqr' command and the resulting feedback gain matrix was $K^T = [-324.3 - 34.20.0 - 0.106]$.

Chapter 3

Methodology

3.1 Modelling the motorcycle

When modeling the motorcycle, the basis given in the Arduino Engineering Kit was used. This kit consists of the motorcycle, equations of motion for balancing and a start has been made on a Simulink model.

3.1.1 Motorcycle

The Engineering Kit consists of various parts with which a motorcycle can be built. These are connected to an Arduino controller which controls the various motors. Figure 3.1 shows the motorcycle.

Figure 3.1: Figure shows the motorcycle[1]

3.1.2 Equations of motion

The equations of motion for balancing the motorcycle are detailed explained in Appendix A. The Simulink model was created, based on, using equation A.19:

$$\dot{x}(t) = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \dot{\theta} \\ \frac{gsin(\theta)()m_r l_{AB} + m_w l_{AC} - \tau_m}{m_w(0.5R^2 + l_{AC}^2) + 0.086m_r(3r^2 + l_{AD}^2) + m_r l_{AB}^2} \\ \tau_m / I_w^C - \ddot{\theta} \end{bmatrix} \tag{3.1}$$

The motorcycle must also be able to balance while cornering. The lean angle during cornering is never more than 90 degrees to the left or right side. In addition, tests show that the inertia wheel cannot adjust by more than 25 degrees. Formula 4.5 for the lean angle has been drawn up for this purpose:

$$\phi_s = \frac{v_s^2}{r_s * g} \tag{3.2}$$

where ϕ_s is the bias term used addition to the lean angle. v_s is the speed of the rear wheel of the motor in $[m/s]$, r_s is the radius of the corner of the motor in upright position used in [m] and g as gravity (this will be 9.81 $[m/s^2]$ on earth].

3.1.3 Simulink model

Hardware model 6 (taken from the Arduino website) was used as the basis for the motor cycle model. The simplified view of this model can be seen in figure 3.2. As shown in this figure, the model is divided into a "controller" and a "plant". The plant is developed in the Motorcycle sub-model (figure X), which contains the engines and the feedback from the various sensors that are present. The control is detailed in the sub-model of the controller. This sub-model consists of:

- a list of standard conditions that the engine must meet in order to be allowed to start at all (figure 3.5).
- Preprocessing of the sensor values (figure 3.6).
- Feedback gains (figure 3.7).

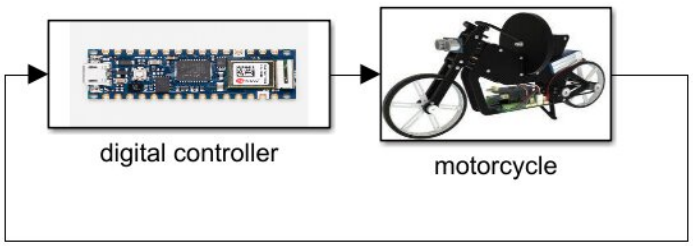

Figure 3.2: Figure shows the model of motorcycle and controller

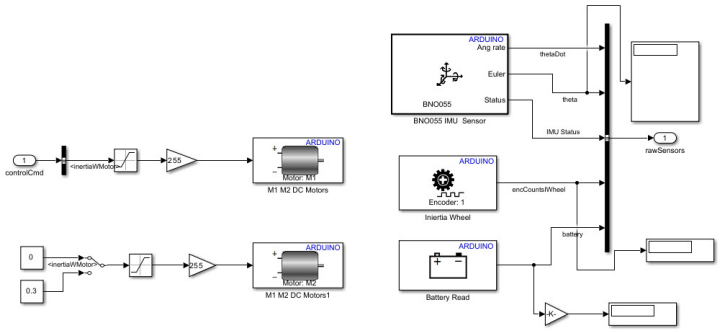

Figure 3.3: Figure shows the model of the motorcycle

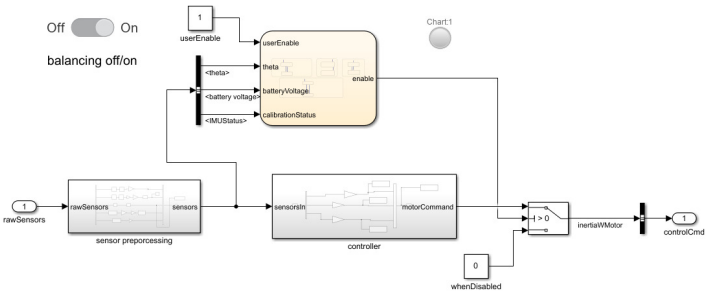

Figure 3.4: Figure shows the model of the controller

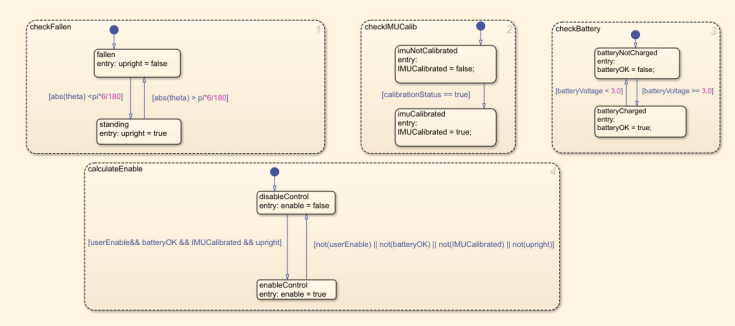

Figure 3.5: Figure shows the chart of basic instructions

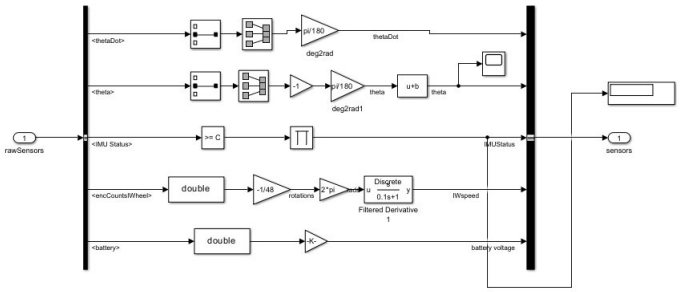

Figure 3.6: Figure shows the pre processing model of the controller

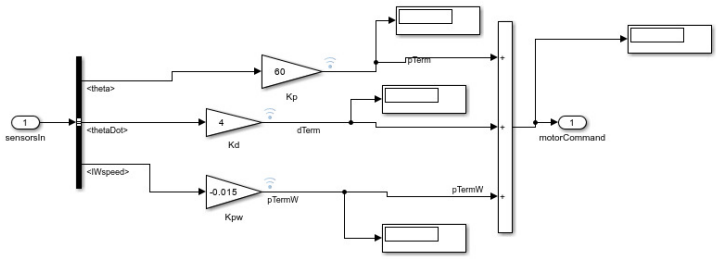

Figure 3.7: Figure shows the feedback gains of the controller

3.2 System Linearization

From the equations of motion of the bike A.20, at the equilibrium point $(\theta, \dot{\theta}, \omega) = (0, 0, 0)$ and substituting $sin(\theta) \approx \theta$.

Finding the Jacobian matrices A and B:

$$A = \begin{pmatrix} \frac{\partial \dot{\theta}}{\partial \theta} & \frac{\partial \dot{\theta}}{\partial \dot{\theta}} & \frac{\partial \dot{\theta}}{\partial \omega} \\ \frac{\partial \ddot{\theta}}{\partial \theta} & \frac{\partial \ddot{\theta}}{\partial \dot{\theta}} & \frac{\partial \ddot{\theta}}{\partial \omega} \\ \frac{\partial \dot{\omega}}{\partial \theta} & \frac{\partial \dot{\omega}}{\partial \dot{\theta}} & \frac{\partial \dot{\omega}}{\partial \omega} \end{pmatrix} \tag{3.3}$$

Therefore,

$$A = \begin{pmatrix} 0 & 1 & 0 \\ \frac{g(m_r l_{AB} + m_\omega l_{AC})}{I_\omega^A + I_r^A} & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{3.4}$$

And,

$$B = \begin{pmatrix} \frac{\partial \dot{\theta}}{\partial \tau_m} \\ \frac{\partial \ddot{\theta}}{\partial \tau_m} \\ \frac{\partial \dot{\omega}}{\partial \tau_m} \end{pmatrix} \tag{3.5}$$

Therefore,

$$B = \begin{pmatrix} 0 \\ -1/\left(I_\omega^A + I_r^A\right) \\ 1/I_\omega^C \end{pmatrix} \tag{3.6}$$

Now, from the parameters of the self-balancing bike, the matrices A and B become:

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 93.5465 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \& B = \begin{pmatrix} 0 \\ -338.406 \\ 11515.3 \end{pmatrix} \tag{3.7}$$

Therefore, we have the state space representation of the linearized system in the form: $\dot{x} = Ax + Bu$ where, $u = -kx$

where, k is the feedback gain matrix, which can be computed from methods like pole placement and LQR (Linear Quadratic Regulator).

3.3 Pole placement

To linearize the state-space equation, since the lean angle θ is small, $\sin(\theta) = \theta$. From the equation A.20, the equation is transformed to type:

$$\dot{x} = Ax(t) + Bu(t)$$
$$y = Cx(t) + Du(t)$$

with

$$\begin{aligned} A &= \begin{bmatrix} 0 & 1 & 0 \\ \frac{g(m_r l_{AB} + m_w l_{AC})}{m_w(0.5R^2 + l_{AC}^2) + 0.086 m_r(3r^2 + l_{AD}^2) + m_r l_{AB}^2} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 93.6838 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ B &= \begin{bmatrix} 0 \\ -\frac{1}{m_w(0.5R^2 + l_{AC}^2) + 0.086 m_r(3r^2 + l_{AD}^2) + m_r l_{AB}^2} \\ -\frac{1}{I_w^C} \end{bmatrix} = \begin{bmatrix} 0 \\ -339 \\ 11511 \end{bmatrix} \\ C &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \\ D &= 0 \end{aligned} \tag{3.8}$$

The schematic of a full-state feedback system is shown in 3.8

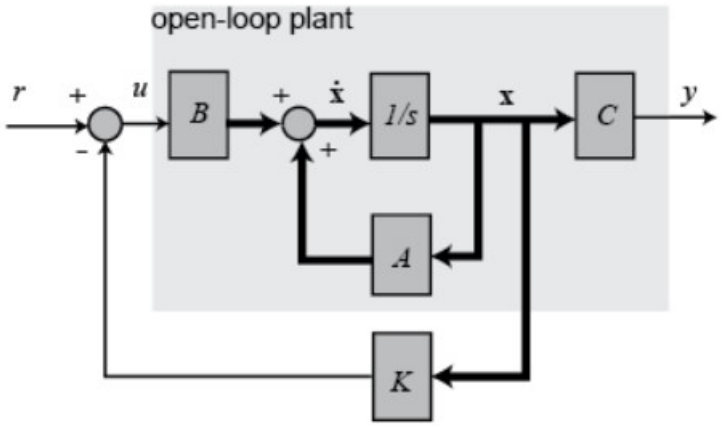

Figure 3.8: Schematic of pole placement approach [2]

The pole of the system state space is the root of the characteristic equation given by $|sI - A| = 0$. Full-state feedback is utilized by commanding the input vector u = -Kx. Substituting into the state space equation above, the equation becomes:

$$\dot{x} = (A - BK)x$$

$$y = (C - DK)x$$

Before attempting this method, where to place the closed-loop poles needs to be decided. Suppose the criteria for the controller were settling time < 0.1 and overshoot $< 10\%$ [7].

The poles of the closed-loop are $s_{1,2} = -\zeta\omega_n \pm \omega_n\sqrt{1-\zeta^2}$. Based on the conditions above:
Settling time: $T_s = \frac{4}{\zeta\omega_n} < 0.1 \leftrightarrow \zeta\omega_n > 40$
Overshoot: $\%OS = e^{-\frac{\zeta\pi}{\sqrt{z-zeta^2}}}.100 < 10\% \leftrightarrow \omega_n\sqrt{1-\zeta^2} > 54$

Combining the two equations and practical considerations, the optimally designed poles should have a real part ranging from 50 onwards in step 5. The imaginary part should vary from 55 onwards. Additionally, the third pole can be positioned between -0.001 and -1, as it is sufficiently slow and will have minimal impact on the inertia wheel speed response.

3.4 Linear Quadratic Regulator (LQR)

Given the linearized state space representation:

1. Defining the LQR (Linear quadratic regulator) cost function: The LQR controller will minimize the cost function:

$$J = \int_0^\infty \left(x^T Q x + u^T R u\right) dt \tag{3.9}$$

   where, Q and R are weighting matrices, found using the Bryson's method. These matrices determine the trade-off between state deviations and control effort.

2. Determining Q and R matrices:

   Q is a diagonal matrix of size equal to the number of states of the system. The values of the diagonal are determined as the reciprocal of the square of the maximum allowable value of

each state variable respectively. Similarly, R is determined by taking the reciprocal of the maximum allowable value of the control input.

Therefore, the values for Q and R can be calculated using the maximum allowable values of the state variables and the control input as given below:

$\theta_{max} = 0.0468rad$ (2.679 deg)

$\dot{\theta}_max = 2rad/s$

$\omega_{max} = 300rad/s$ (7370 rpm)

$\tau_{max} = 0.0265Nm$ (270 gcm)

3. Continuous-Time Algebraic Riccati Equation (CARE):

   To find the gain matrix k, we have to solve the CARE equation given below:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{3.10}$$

   The solution P to this equation will help us compute the gain matrix k:

$$K = R^{-1}B^T P \tag{3.11}$$

4. Computing the LQR gain matrix k: This was done in MATLAB using the 'lqr' command.

   Different feedback gain matrices can be found by determining the Q and R values for the LQR controller using the Bryson's rule, and then tweaking them by testing them on the self-balancing bike and also checking the Lyapunov stability of the system using the method given in the next section.

3.5 Lyapunov Stability

To verify stability using the Lyapunov function, the following function is used:

$$V(x) = x^T Px \tag{3.12}$$

The time derivative of $V(x)$ along the trajectories of the system is:

$$\dot{V}(x) = \frac{d}{dt}\left(x^T Px\right) = \dot{x}^T Px + x^T P\dot{x} \tag{3.13}$$

Using the system dynamics $\dot{x} = (A - BK)x$:

$$\dot{V}(x) = x^T (A - BK)^T Px + x^T P(A - BK)x \tag{3.14}$$

$\dot{V}(x)$ should be negative definite for the system to have stability. Therefore, for $\dot{V}(x)$ to be negative definite:

$$(A - BK)^T P + P(A - BK) < 0 \tag{3.15}$$

The matrix $(A - BK)^T P + P(A - BK)$ is negative definite if and only if all the eigenvalues have negative real parts.

3.6 Discretization

When designing a control system for a practical application, such as the self-balancing bike, it's important to convert the continuous-time controller to discrete-time controller that can be implemented in the digital system. Therefore, discretizing the LQR controller is crucial for the real-time application on the self-balancing bike where the controller is implemented using the Arduino Nano 33 IOT microcontroller.

The discrete-time state-space representation with sampling time Ts is:

$$x[k+1] = A_d x[k] + B_d u[k] \tag{3.16}$$

where, the discretized system matrices A_d and B_d can be found using the 'c2d' command in MATLAB with zero-order hold (ZOH) method.

After this the same Q and R matrices found using the Bryson's rule in the previous section can be used to compute the gain matrix k_d using MATLAB's 'dlqr' function.

Chapter 4

Results

4.1 Modelling the motorcycle

The updated Simulink model of the motorcycle was built based on the steps below. The updated Simulink can be found in the next section.

1. Selected the basic model (Hardware model 6 from the Arduino Engineering Kit) and then connected the engines and encoders to the correct ports of the controller.
2. Added straight forward movement: motor and encoder (each revolution on the motor shaft produces 3 pulses) in the Simulink model with the control (on/off) in the controller. The speed of the rear wheel can be calculated because there is an encoder on the motor:

$$v_s = v_m * gb * c_{rw} \tag{4.1}$$

   Formula 4.1 is needed in one of the next steps to calculate the bias. With v_s as speed of the rear wheel. With g_b as ratio of the gearbox, this will be 1/100 [3]. v_m is the speed of the output shaft on the motor in [rad/s]. c_{rw} will be the circumference of the rear wheel. This can be calculated by equation 4.2.

$$c_{rw} = \pi * 2 * r_{rw} \tag{4.2}$$

   With r_{rw} as radius of the rear wheel, the radius will be 4.5 [cm].

3. Added movement for steering: motor in the Simulink model with the control (left/right) in the controller. The cornering circle can be calculated based on the angle at which the handlebars of the motor cycle are positioned, see equation 4.4.

$$R = \frac{W}{tan(\alpha)} \tag{4.3}$$

$$r_s = \sqrt{\frac{W}{2}^2 + R^2} \tag{4.4}$$

   w is the length of the motor from shaft to shaft, this will be 23 [cm], and α is the angle of the handlebars of the motorcycle, and R as distance from rear wheel until center of cornering circle. r_s shows the radius of the cornering circle. Equations 4.3 and 4.4 are based on the schematic representation of the cornering circle at figure 4.1.

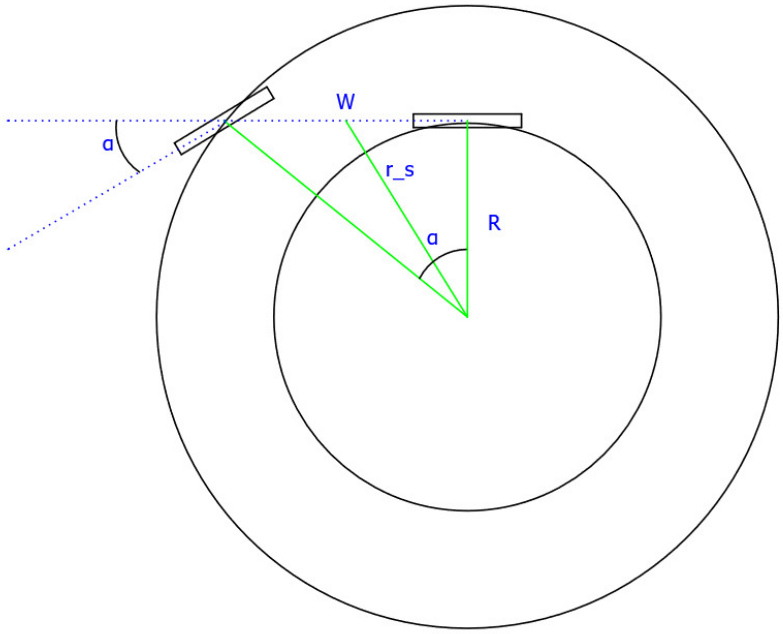

Figure 4.1: Cornering circle of the motor cycle

4. Bias adjustment for balancing. The motor cycle tilts in the opposite direction when it starts to steer, a bias has been added to change the equilibrium angle. This is possible using this calculation:

$$\phi_s = \frac{v_s^2}{r_s * g} \tag{4.5}$$

   Next to the bias adjustment when the motor cycle tilts in the opposite direction, there is a bias added of -1.3 degrees as compensation for an incorrect measurement in the IMU sensor.

5. As a final step, the state feedback gains for balancing were adjusted. In the following subchapters a number of different ways of determining state feedback gains values are discussed.
   The feedback sign of gains $k_{p\omega}$ is changed to negative.

4.2 Simulink model

This section describes the changes made to the given Simulink model as described in chapter 3.1.3.

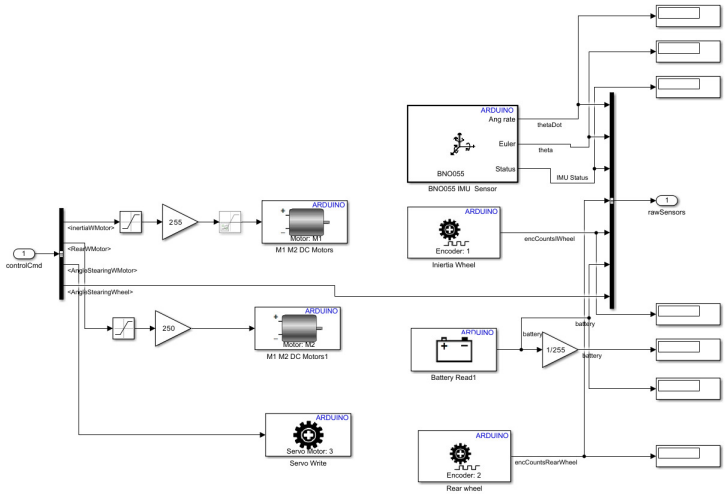

Figure 4.2: Figure shows the updated model of motorcycle and controller

Figure 4.2 shows the modified engine model. The ports of the motors and encoders have been adjusted. In addition, the motor with encoder for steering has been added.

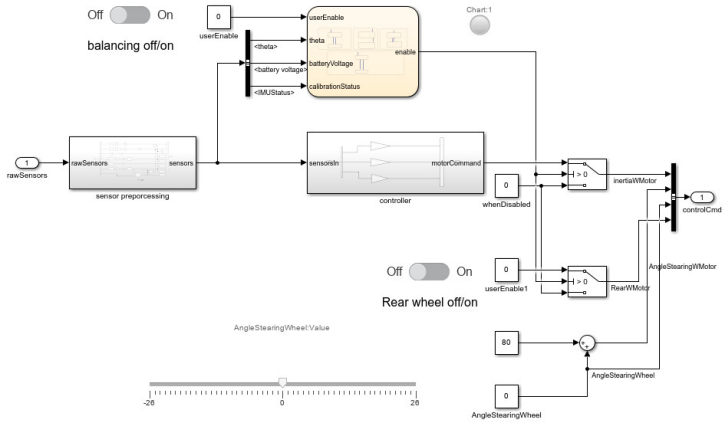

Figure 4.3: Figure shows the updated model of the controller

Figure 4.3 shows a modified model of the controller. In this model the possibility of steering the engine has been added and starting/stopping the engine for the rear wheel.

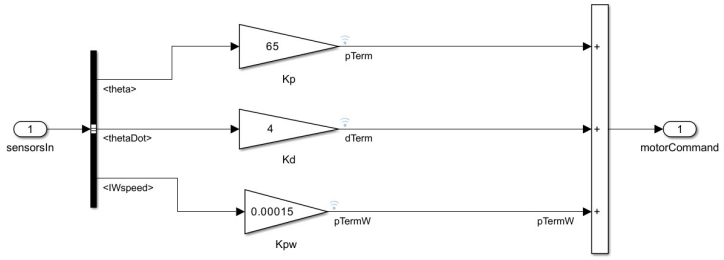

Figure 4.4: Figure shows place of the controller to update the feedback gains

Figure 4.4 shows the position at the controller to change the feedback gains. Unlike the original model, the feedback sign of gains $k_{p\omega}$ is changed to negative. Note: this figure doesn't show the best fitted gains.

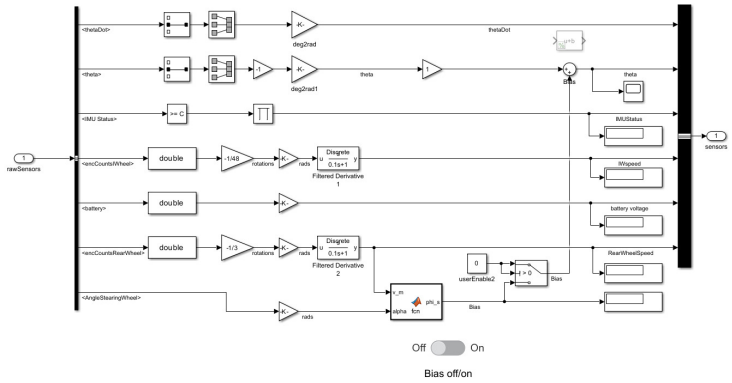

Figure 4.5: Figure shows the updated pre processing model of the controller

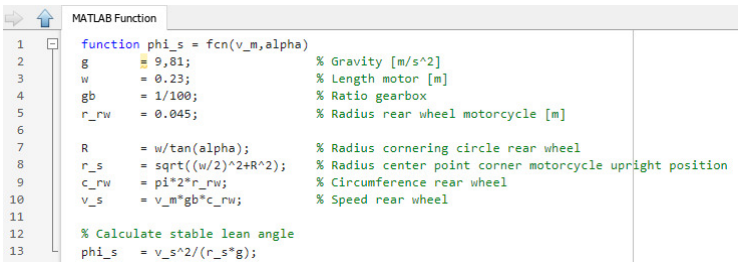

Figure 4.6: Figure shows the bias for the updated pre processing model of the controller

Figure 4.5 shows the updated model of the pre processing part of the sensors. At this spot there is a bias added of -1.3 degrees. In this part of the controller the adjustment for moving the equilibrium point while the motor cycle is steering has been added. Figure 4.6 shows how the equilibrium point is adjusted as the bias is adjusted.

4.3 Pole placement

One of the first things that needs to be analyzed is whether the open-loop system is stable. The eigenvalue of matrix A is $\begin{bmatrix} 9.6790 & -9.6790 & 0 \end{bmatrix}$. As you can see one of the poles is in the right half plane which means that the open-loop system is unstable. A non-zero initial condition is added to observe what happens to this unstable system. The plot is shown in Figure 4.7.

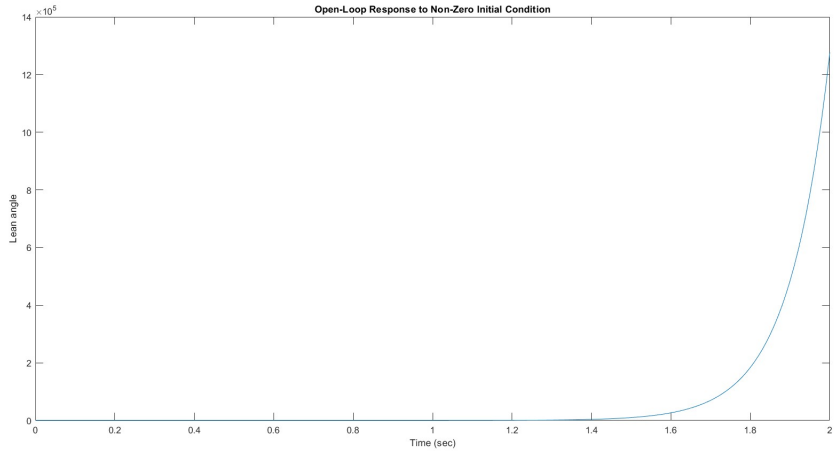

Figure 4.7: Open-loop system

The next step is checking whether the system is controllable or not. With the help of Matlab, it is easy to check that the system is controllable. As mentioned before, the designed poles range from right to further to left to determine the optimal poles. Since there are too many efforts to test, this part only displays 4 results.

First, we try the first 2 poles are -50 $\pm$ 55j, and the third pole runs from -0.001 to -1. The result is shown below:

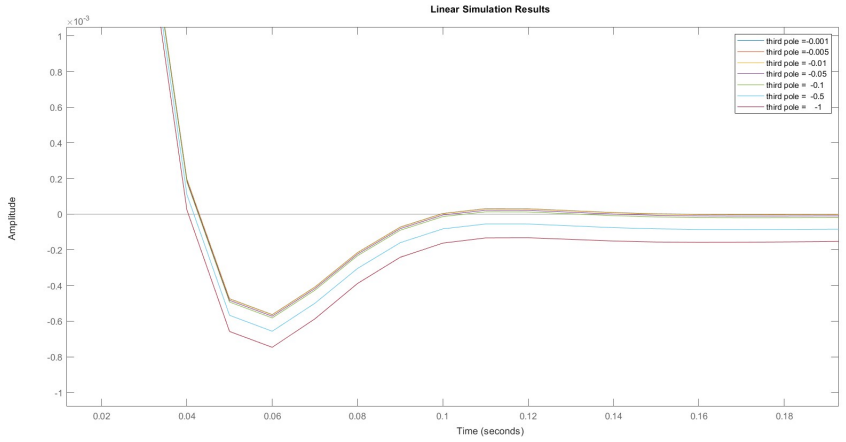

Figure 4.8: Lean angle of the motorcycle in first experiment

Second, we try the first 2 poles are -70 ± 100j.

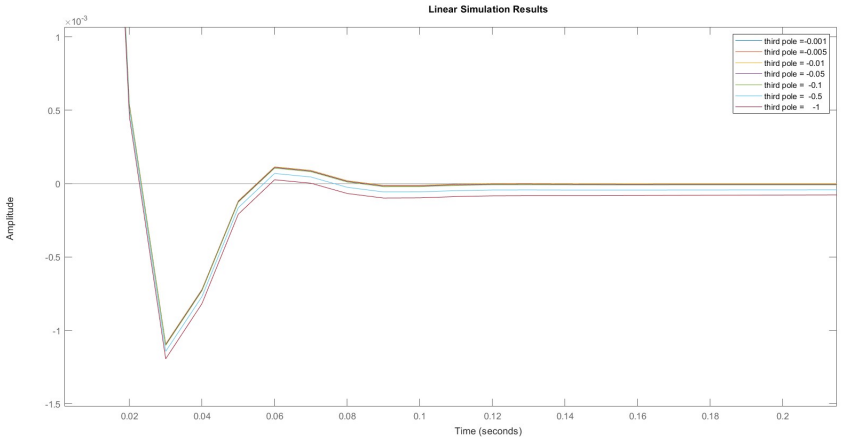

Figure 4.9: Lean angle of the motorcycle in the second experiment

Next, we try the first 2 poles are -75 ± 125j.

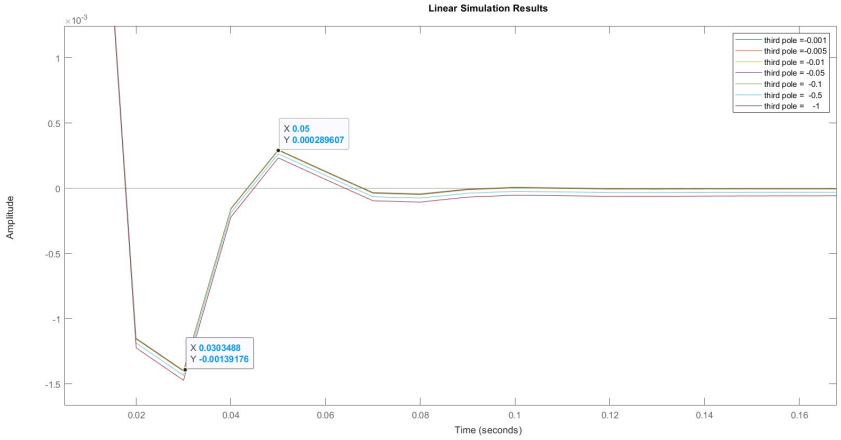

Figure 4.10: Lean angle of the motorcycle in third experiment

Finally, we try the first 2 poles are -100 ± 125j.

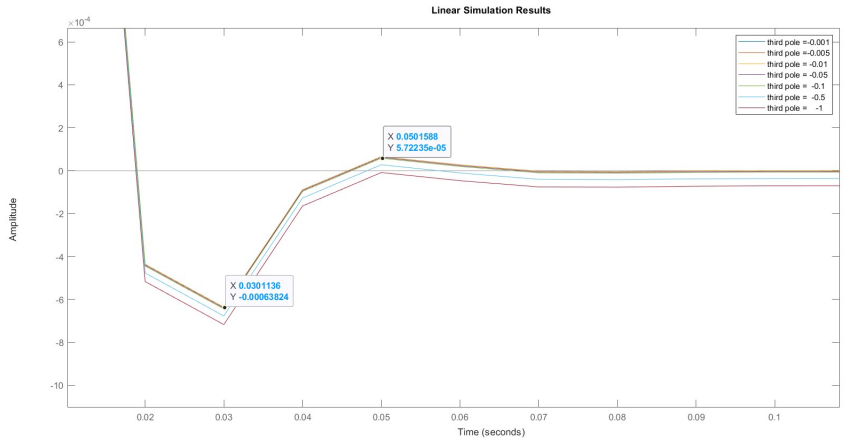

Figure 4.11: Lean angle of the motorcycle in fourth experiment

As seen in the figure, the farther the designed poles are to the left, the less overshoot occurs, which requires more control effort. However, after testing the results on the real system, the motorcycle could not balance. Therefore, based on those gains and multiple tests, we adjusted the gains to achieve the best balance position. The gains we obtained are: $\begin{bmatrix}60 & 4 & 0.015\end{bmatrix}$. To check with the pole placement method, the designed poles are $\begin{bmatrix}-1.1635 & -0.0165 & -0.0008\end{bmatrix}$. Since there is one pole equals zero, the system is asymptotic stable.

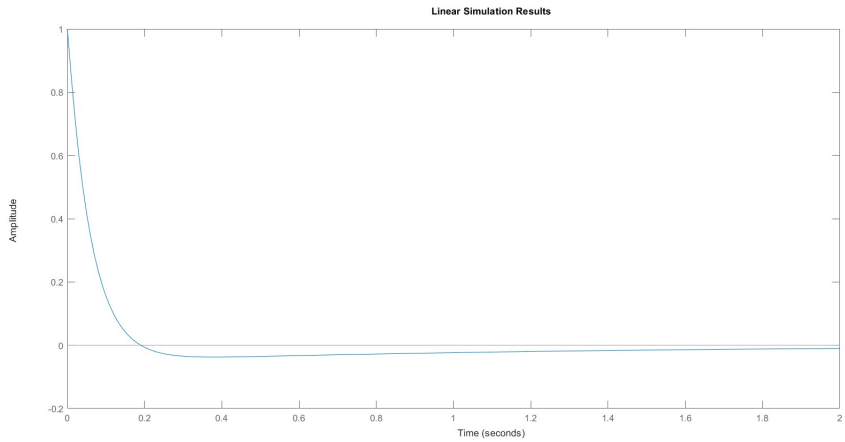

Figure 4.12: Lean angle of the motorcycle in adjustment experiment

In Figure 4.12, the signal starts from the initial angle (which is set to nonzero), decreases to approximately 0 after 0.2 seconds then slightly increases. Compared with the real system which is shown in Figure 4.13, the motorcycle could balance well for around 3 minutes.

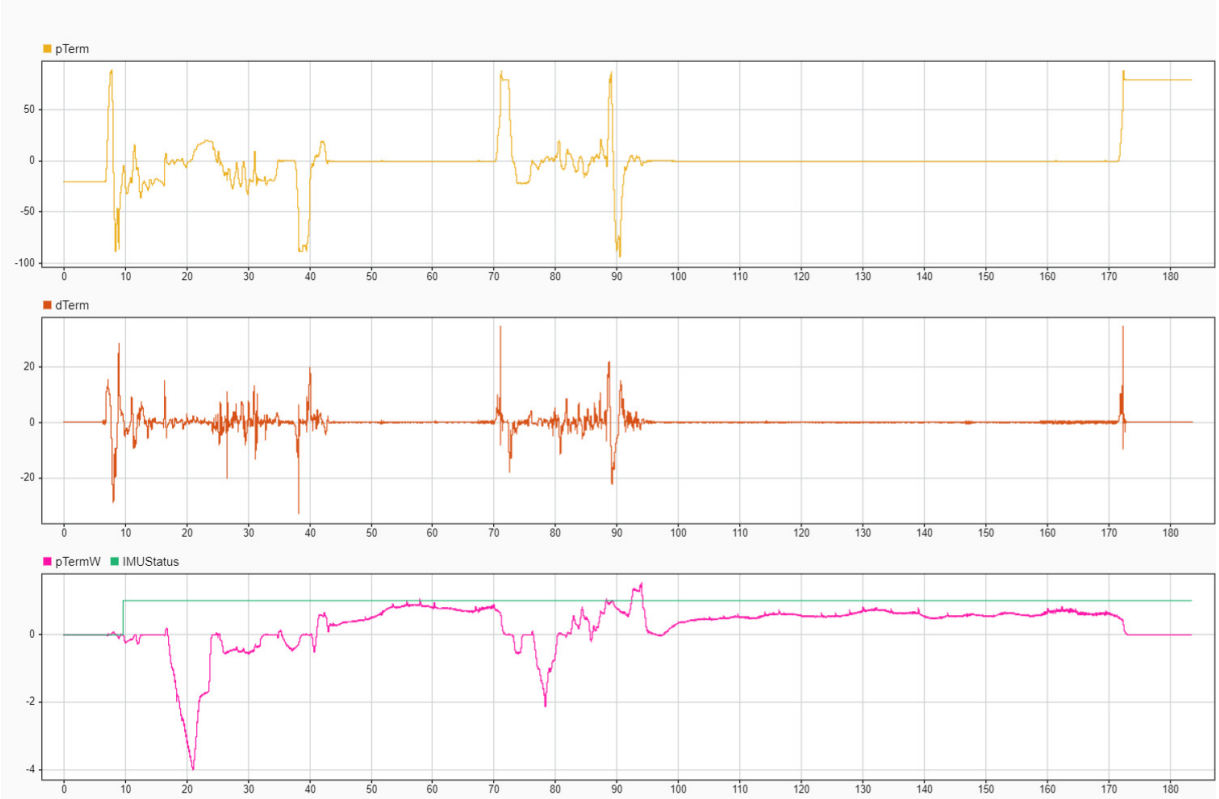

Figure 4.13: Log signals of real system

4.4 Linear Quadratic Regulator (LQR)

The LQR controller as discussed in the methodology part of this report was computed in MATLAB while using the Bryson's rule for determining the values for Q and R.

Therefore, Q and R were found to be:

$$Q = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 0.2500 & 0 \\ 0 & 0 & 1.111e-5 \end{bmatrix}$$

$$R = 1.424e03$$

And then using 'lqr' command from MATLAB, the gain matrix k was found to be:

k = [-0.7071, -0.0691, -0.0001]

These gain values were then tested on the state space model of the self-balancing bike in Simulink, the model was simulated for 10 seconds with initial value of the lean angle as 10 degrees and the following result was achieved.

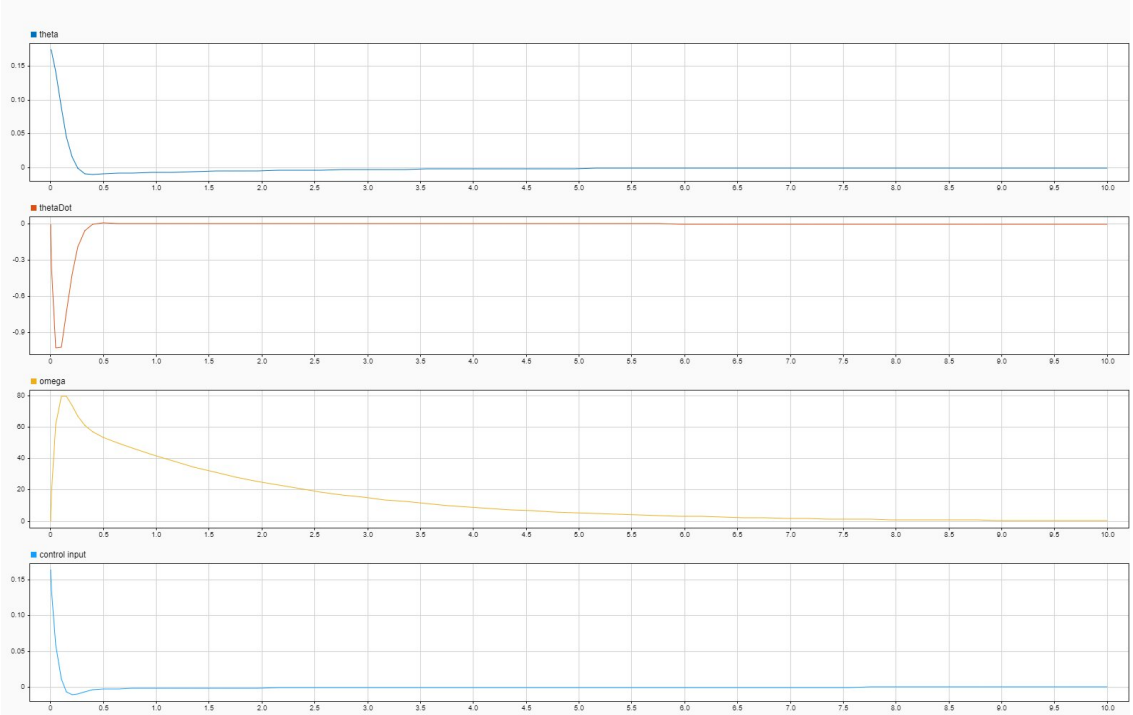

Figure 4.14: LQR state space model results

The lean angle θ of the bike comes back to the equilibrium point in around 0.4 seconds.

Below is a table of results from LQR with different Q and R values and the Lyapunov stability of the system with the resulting state feedback gains.

Note: The last row shows the results of using the Bryson's rule to calculate feedback gains.

Q(3x3)	R	k	Eigenvalues (real parts)	Lyapunov Stable
Identity Matrix	1	(-658.8, -68.12, -1.0)	-0.0010 ; -0.0010 ; -1.1520	Yes
Identity Matrix	0.5	(-932.5, -96.32, -1.41)	-0.0010 ; -0.0010 ; -1.6292	Yes
diag(1,1,0.5)	1	(-466.2, -48.2, -0.071)	-0.0097 ; -0.0097 ; -8.1496	Yes
diag(100, 0.25, 1.11e-5)	1.424e03	(-0.7071, -0.0691, -0.0001)	-0.4455 ; -12.6017 ; -12.6017	Yes

Table 4.1: Table of Eigenvalues and Stability Results

4.5 Discretized LQR controller

As discussed on the methodologies section, discretization of the LQR controller is very important while implementing it on the Arduino microcontroller on the self-balancing bike.

Hence, using the 'c2d' and 'dlqr' commands of MATLAB the feedback gains were found and tested on the bike. But these gains were found to generate not strong enough control actuation for proper balancing. Hence, different values of Q and R outside of the Bryson's rule were also tried which resulted into an unstable system looking at the eigenvalues.

Note: The last row shows the results of using the Bryson's rule to calculate feedback gains.

Results:

Q(3x3)	R	k	Eigenvalues (real parts)
Identity Matrix	1	(-5.7, -0.59, -0.0079)	0.9078 ; 0.9078 ; 0.0001
Identity Matrix	0.5	(-5.7, -0.59, -0.0079)	0.9079 ; 0.9079 ; 0.0002
diag(1,1,0.5)	1	(-5.7, -0.59, -0.0079)	0.9079 ; 0.9079 ; 0.0002
diag(100, 0.25, 1.11e-5)	1.424e03	(-0.8621, -0.0737, -0.0001)	0.9956 ; 0.8792 ; 0.8792

Table 4.2: Table of Eigenvalues and Stability Results

4.6 WIFI connection

After determining the optimal gains for self-balancing the motorcycle, the next step is to test its straight motion. However, since the model is operated in external mode via a USB cable, the motorcycle must stay close to the laptops, causing the cable to touch the wheel and disrupt the smooth operation. Therefore, before testing straight motion, a wireless communication protocol, such as Wi-Fi, needs to be configured to allow the motorcycle to drive away from the computers. Steps needed to configure a model to run via Wifi could be found in [1].

For Wi-Fi configuration, the laptop and Simulink model must share the same Wi-Fi properties. However, this configuration is not stable. Initially, after unplugging the USB cable, the Wi-Fi connection works well, but it disrupts after approximately 30 to 60 seconds. The Figure 4.15 shows that error.

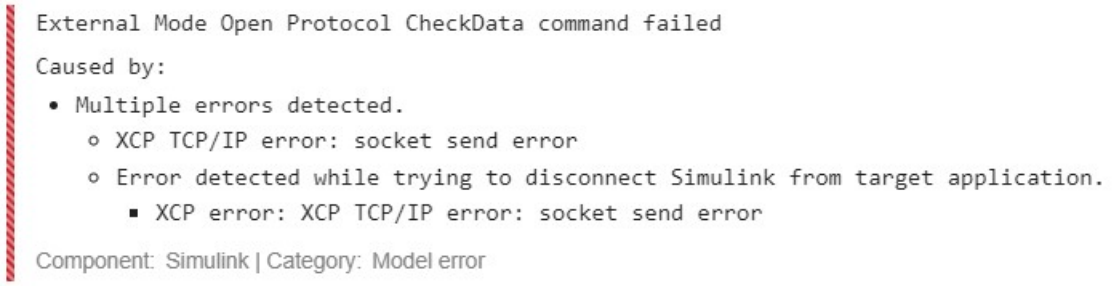

Figure 4.15: XCP error

4.7 Results conclusion

With the gains found from the Pole placement method and the LQR controller the bike could not balance well, therefore, after conducting more tests on the real system, based on those gains and multiple tests, we adjusted the gains to achieve the best balance position. The gains we obtained are: $\begin{bmatrix} -60 & -4 & -0.015 \end{bmatrix}$. The eigenvalues for these gains all have negative real parts: -1.1635; -0.0165; -0.0008, this shows that the system is stable.

Chapter 5

Conclusion

In this project the design and implementation of a self-balancing motorcycle using advanced control strategies was explored.

The approach began with fixing the Simulink model provided from the Arduino website and incorporating the blocks for straight and cornering motion of the bike. The bias adjustment for the lean angle θ was a critical step in ensuring the bike's stability.

Through various pole placement experiments it was found that placing the poles farther to the left reduced the overshoot but required more control effort. Ultimately, the feedback gains were adjusted to [-60, -4 , -0.015], resulting a system that could balance for approximately 3 minutes.

The LQR (Linear Quadratic Regulator) controller was implemented using Bryson's rule to determine the values for Q and R. The optimal feedback gains obtained were [-0.7071, -0.0691, -0.0001], which brought the lean angle back to equilibrium within 0.4 seconds in simulations on the state space model of the bike. And the gains found from the discrete LQR were found to generate weak control actuations.

However, further refinements were necessary for real-world application, leading to the best-performing gains finally chosen to be implemented on the bike: [-60, -4, -0.015]. With these gains the bike could continuously balance for 2-3 minutes.

Additionally, the wireless communication was tried for the bike to control it during straight and cornering motion. The WIFI configuration had revealed errors, with connections failing after 30 to 60 seconds with an 'XCP connection error'.

Overall, the findings from this project demonstrate that with precise tuning and appropriate control strategies, a self-balancing motorcycle can achieve short-term stability. Future work should focus on improving wireless communication reliability and exploring alternative control methods to enhance long-term performance and robustness. Further experimentation with different Q and R values and advanced control techniques may yield even better results for both stability and operational duration.

Chapter 6

Recommendations

The subjects that are recommended to be addressed for improvements are listed below:

1. When looking for correct feedback gains, it seems that the response of the motorcycle to the gains depends on the laptop. The reason for this difference can be investigated in a subsequent study.
2. The WIFI module gives an XCP error when loading the software on the Arduino. This needs to be explored.
3. Driving straight ahead and balancing was not successful during this project, this still needs to be addressed.
4. After a few minutes of operation, the IMU sensor displays an error and briefly displays its initialized value. This needs to be explored.
5. The motorcycle is currently balancing without causing many disturbances. In the future, the next assessment can improve this.

List of Figures

List of Tables

Bibliography

[1] Self-Balancing Motor Cycle using Arduino Engineering Kit Rev 2 - MATLAB Simulink - MathWorks Benelux.

[2] State-space methods for controller design.

[3] TT MOTOR (HK) INDUSTRIAL CO.LTD. NOMINAL MOTOR CURVES. Technical report.

[4] Selda Guney and Ayten Atasoy. An approach to pole placement method with output feedback. In *UKACC Control Conference*, 2011.

[5] Michael A Henson and Dale E Seborg. Feedback linearizing control. *Nonlinear process control*, 4:149–231, 1997.

[6] Jean-Jacques E Slotine. Applied nonlinear control. *PRENTICE-HALL google schola*, 2:1123–1131, 1991.

[7] Adam Wonia, Michał Wonia, and Robert Piotrowski. *Control of a Nonlinear and Linearized Model of Self-balancing Electric Motorcycle*, pages 360–371. 01 2020.

Appendix A

Equations of motion motorcycle

This chapter describes the equations of motion of the motorcycle, which are given in the Arduino engineering kit [1]. The motorcycle can be seen as an inverted pendulum because the motorcycle has only one point on the ground (see point A in figure A.1) and the inertia wheel is connected to the motorcycle with one degree of freedom. When drawing up these equations, it was assumed that the motorcycle can only weigh freely over the wheel-ground axis (an imaginary line that connect the points on the rear and front wheels where touch by the ground), the thickness of the wheels and the friction between moving parts is negligible.

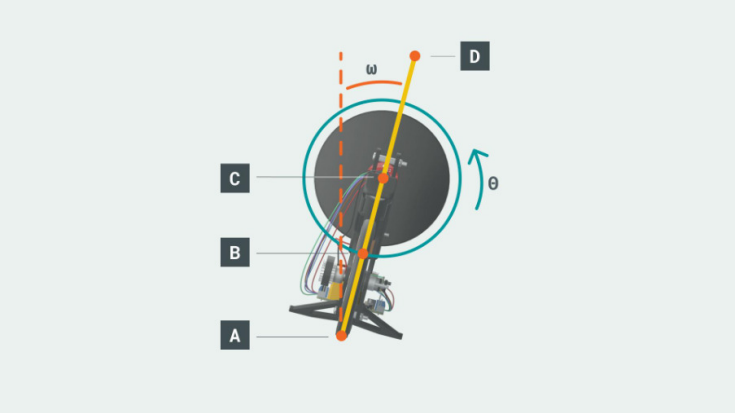

Figure A.1: Schematic drawing of the motorcycle [1]

Figure A.1 is a schematic representation of the motorcycle. In contrast to the figure, θ and ω must be reversed. From the numbered points at the figure some important parameters can be made such as l_{AD} as the length of the pendulum rod, the center of the mass will be point at point B and point C is the rotation axis of the inertial wheel. In addition to the parameters mentioned above, θ is also visible as lean angle in degrees, at an angle of 0 degrees the motorcycle is perfectly upright. Also ω is visible as rotational speed and $\dot{\omega}$ as acceleration.

$$F = mA \tag{A.1}$$

with F as the force in [N], m as the mass in [kg] and a as acceleration in $[m/s^2]$.

$$\tau = I\ddot{\theta} \tag{A.2}$$

with τ as the torque in [Nm], I as the moment of inertia in kg/m^2 and $\ddot{\theta}$ as angular acceleration in rad/s^2.

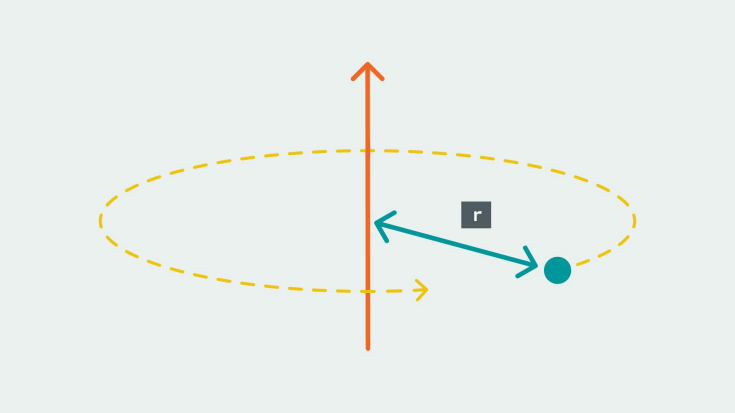

Figure A.2: Schematic drawing of the point mass [1]

A point mass (see figure A.2) doesn't have a moment of inertia on its own axis. It uses the parallel axis theorem, this is showed in the next equation.

$$I = Mr^2 \tag{A.3}$$

with point mass M in [kg] at distance r in [mm].

Solid disk radius (see figure A.3) is a special case of a solid cylinder, with h=0: consequence of the perpendicular axis theorem.

$$I_z = 0.5mr^2 \tag{A.4}$$

$$I_x = I_y = 0.25mr^2 \tag{A.5}$$

both with mass m in [kg] at distance r in [mm].

The solid cylinder (see figure A.4) is a thick walled cylinderical tube.

$$I_z = 0.5mr^2 \tag{A.6}$$

$$I_x = I_y = 0.086m(3r^2 + h^2) \tag{A.7}$$

both with mass m in [kg] at distance h and r in [mm].

$$\tau_{gr} = m_r g l_{AB} sin(\theta) \tag{A.8}$$

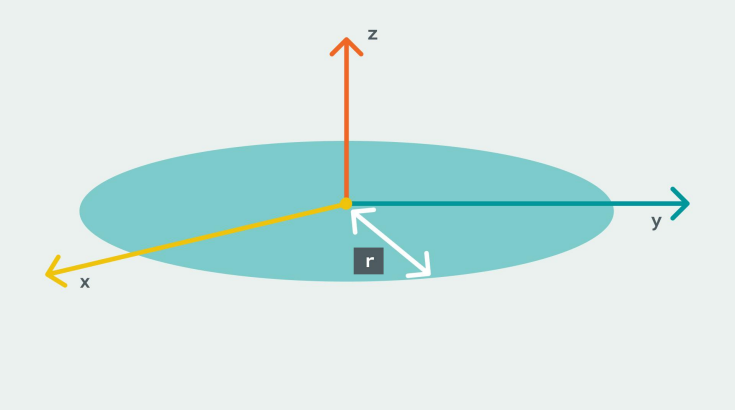

Figure A.3: Schematic drawing of the solid disk radius [1]

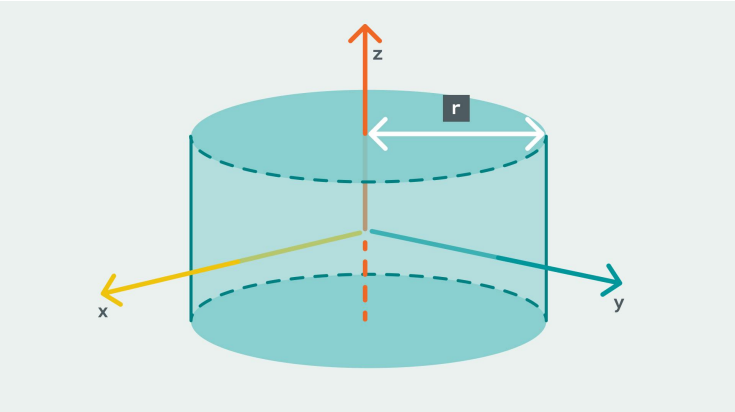

Figure A.4: Schematic drawing of the solid cylinder radius [1]

with torque due gravity applied to the pendulum rod τ_{gr}, mass of the rod m_r in [kg], gravitational acceleration g, distance between points A and B l_{AB} in [mm].

$$\tau_{gw} = m_w g l_{AB} sin(\theta) \tag{A.9}$$

with torque due gravity applied to the inertia wheel τ_{gw}, mass of the inertia wheel m_w in [kg], gravitational acceleration g, distance between points A and B l_{AB} in [mm].

$$I_w^C = 0.5 m_w R^2 \tag{A.10}$$

I_w^C calculates the moment of inertia wheel about point C. To calculate the moment of inertia around point A I_w^A the parallel axis theorem will be used.

$$I_w^A = I_w^C + m_w l_{AC}^2 = 0.5 m_w R^2 + m_w l_{AC}^2 = m_w (0.5 R^2 + l_{AC}^2) \tag{A.11}$$

I_w^A will also apply the parallel axis theorem. This states the body's moment of inertia about any given axis is the moment of inertia about the center of mass plus the mass of the body times the distance between the point and the center of mass squared.

$$I_r^B + 0.086 m_r (3r^2 + l_{AD}^2) \tag{A.12}$$

$$I_r^A = I_r^B + m_r l_{AB}^2 = 0.086 m_r (3r^2 + l_{AD}^2) + m_r l_{AB}^2 \tag{A.13}$$

Because the pendulum rotates around point A and not around point B, the parallel axis theorem is again applied.

The following steps describes the equation of motion:

$$\tau = I\ddot{\theta} \tag{A.14}$$

$$\tau_{gr} + \tau_{gw} + \tau_m = (I_w^A + I_r^A)\ddot{\theta} \tag{A.15}$$

$$m_r g l_{AB} sin(\theta) + m_w g l_{AC} sin(\theta) - \tau_m = (m_w (0.5R^2 + l_{AC}^2) + 0.086 m_r (3r^2 + l_{AD}^2) + m_r l_{AB}^2)\ddot{\theta} \tag{A.16}$$

$$\tau_m = I_w^C (\dot{\omega} + \ddot{\theta}) \tag{A.17}$$

$$\tau_m = 0.5 m_w R^2 (\dot{\omega} + \ddot{\theta}) \tag{A.18}$$

Next steps are to create state space models of the equations of motion. The first step is to create a state vector (A.19) and the second step is to derive an expression for time derivative of the state vector (**??**).

$$x(t) = \begin{bmatrix} \theta \\ \dot{\theta} \\ \omega \end{bmatrix} \tag{A.19}$$

$$\dot{x}(t) = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \dot{\theta} \\ \frac{gsin(\theta)() m_r l_{AB} + m_w l_{AC} - \tau_m}{m_w (0.5R^2 + l_{AC}^2) + 0.086 m_r (3r^2 + l_{AD}^2) + m_r l_{AB}^2} \\ \tau_m / I_w^C - \ddot{\theta} \end{bmatrix} \tag{A.20}$$

Appendix B

Matlab code

B.1 Pole placement

```
g = 9.80665;          % gravity constant
m_r = 0.2948;         % mass of the rod
m_w = 0.0695;         % mass of the inertia wheel
R = 0.05;             % radius of the inertia wheel
r = 0.02;             % cross section radius of the rod
l = 0.13;             % corresponding lengths
l_AD = l;
l_AC = l;             % assume wheel is mounted on the top of the pendulum
l_AB = l/2;
I_w_C = 0.5*m_w*R^2; % corresponding inertias
I_w_A = I_w_C + m_w*l_AC^2;
I_r_B = (1/12)*m_r*(3*r^2+l_AD^2);
I_r_A = I_r_B + m_r*l_AB^2;
a = (m_r*l_AB + m_w*l_AC)*g;
b = m_w*(0.5*(R^2) + (l_AC^2)) + (1/12)*m_r*(3*(r^2) + (l_AD^2)) + m_r*(l_AB^2);
A = [0 1 0; a/b 0 0; 0 0 0];
B = [0; -1/b; 1/I_w_C];
C = [1 0 0];
D = 0;
sys = ss(A,B,C,D);
eig(A);
ctrb(A,B); rank(ctrb(A,B));
obsv(A,C); rank(obsv(A,C));
t = 0:0.01:2;
u = zeros(size(t));
x0 = [1 0 0];
[y,t,x] = lsim(sys,u,t,x0);
plot(t,y)
title('Open-Loop Response to Non-Zero Initial Condition')
xlabel('Time (sec)')
ylabel('Lean angle')
third_pole = [-0.001;-0.005;-0.01;-0.05;-0.1;-0.5;-1];
for i = 1:1:length(third_pole)
    j = sqrt(-1);
    eigs_1 = [-50+55j;-50-55j;third_pole(i)];
```

```
    K1 = place(A,B,eigs_1)
    sys_cl_1 = ss(A - B*K1,B,C,D);
    figure(1);
    lsim(sys_cl_1,u,t,x0);
    hold on;
end
figure(1)
hold off
legend(strcat('third pole = ',num2str(third_pole)));
for i = 1:1:length(third_pole)
    j = sqrt(-1);
    eigs_2 = [-70+100j;-70-100j;third_pole(i)];
    K2 = place(A,B,eigs_2)
    sys_cl_2 = ss(A - B*K2,B,C,D);
    figure(2);
    lsim(sys_cl_2,u,t,x0);
    hold on;
end
figure(2)
hold off
legend(strcat('third pole = ',num2str(third_pole)));
for i = 1:1:length(third_pole)
    j = sqrt(-1);
    eigs_3 = [-75+125j;-75-125j;third_pole(i)];
    K3 = place(A,B,eigs_3)
    sys_cl_3 = ss(A - B*K8,B,C,D);
    figure(3);
    lsim(sys_cl_3,u,t,x0);
    hold on;
end
figure(3)
hold off
legend(strcat('third pole = ',num2str(third_pole)));
for i = 1:1:length(third_pole)
    j = sqrt(-1);
    eigs_4 = [-100+125j;-100-125j;third_pole(i)];
    K4 = place(A,B,eigs_4)
    sys_cl_4 = ss(A - B*K4,B,C,D);
    figure(4);
    lsim(sys_cl_4,u,t,x0);
    hold on;
end
figure(4)
hold off
legend(strcat('third pole = ',num2str(third_pole)));
K5 = [-60, -4, -0.015];
pole = place(A,B,K5);
sys_cl_5 = ss(A - B*K5,B,C,D);
figure(5);
lsim(sys_cl_5,u,t,x0);
```

B.2 Linear quadratic regulator

Checking controllabilty:

```
% system matrices
A = [0, 1, 0;
     93.5465, 0, 0;
     0, 0, 0];

B = [0;
     -338.406;
     11515.3];

% controllability
controllability_matrix = ctrb(A, B);
rank_of_controllability = rank(controllability_matrix);

if rank_of_controllability < size(A)
    disp('The system is not controllable.');
else
    disp('The system is controllable.');
end
```

Designing the LQR controller:

```
% Define maximum allowable values for states and control input
theta_max = 0.0468;  % radians  2.679 deg
theta_dot_max = 2;  % rad/s
omega_max = 300;  % rad/s ; 7370 rpm
tau_max = 0.0265;  % Nm ; 270 gcm

% % Construct Q and R using Bryson's method
Q = diag([1/theta_max^2, 1/theta_dot_max^2, 1/omega_max^2]);
R = 1/tau_max^2;

% Compute the LQR gain matrix K
[K, S, e] = lqr(A, B, Q, R);

% Display the gain matrix K
disp('The LQR gain matrix K is:');
disp(K);

% Display the solution to the Riccati equation S
disp('The solution to the Riccati equation S is:');
disp(S);

% Display the closed-loop eigenvalues
disp('The closed-loop eigenvalues are:');
disp(e);
```

Checking Lyapunov Stability:

```
%%
% matrix of the closed loop system
A_cl = A - B * K;
eig(A_cl)
```

```
% Lyapunov function matrix P (from the solution to the Riccati equation)
P = S;

Lyap_matrix = A_cl' * P + P * A_cl;

% print the Lyapunov matrix
disp('The matrix A_cl^T * P + P * A_cl is:');
disp(Lyap_matrix);

% eigenvalues of the Lyapunov matrix (should all have negative real parts)
Lyap_eigenvalues = eig(Lyap_matrix);

% print the eigenvalues of the Lyapunov matrix
disp('The eigenvalues of the matrix A_cl^T * P + P * A_cl are:');
disp(Lyap_eigenvalues);

% check if all eigenvalues have negative real parts
is_negative_definite = all(real(Lyap_eigenvalues) < 0);

% print whether the Lyapunov matrix is negative definite or not
if is_negative_definite
    disp('The matrix A_cl^T * P + P * A_cl is negative definite.');
else
    disp('The matrix A_cl^T * P + P * A_cl is NOT negative definite.');
end
```

Discrete LQR Design:

```
%
T_s = 0.01; % Sampling time in seconds

% discretize the continuous-time state-space model
sys_continuous = ss(A, B, [], []);
sys_discrete = c2d(sys_continuous, T_s, 'zoh');

A_d = sys_discrete.A;
B_d = sys_discrete.B;

% weighting matrices Q_d and R_d
Q_d = Q;
R_d = R;

% calculate the discrete-time LQR gain
[K_d, S_d, e_d] = dlqr(A_d, B_d, Q_d, R_d)

disp('Discrete-time LQR gain:');
disp(15*K_d);
```