LUND UNIVERSITY

LTH

FACULTY OF ENGINEERING

Stabilization of the Uni-directional MinSeg robot

By:
Robin Müller, Aslan Hamadi Liafichev
Johanna Häggström Wedding & Ella Thunborg

June 14, 2023

Real-Time Systems
FRTN01

Contents

1 Introduction

The goal of this project is to implement a stabilization controller for a Uni-directional MinSeg Segway. This is a commercial hardware controlled with an Arduino MEGA 2560 R3 that drives a DC motor and reads values from the inertial measurement unit (IMU). The IMU consists of 3 accelerometers and 3 gyroscopes. A rotational encoder is also available for controlling the motor position and speed. The mini robot has restrictions and can only be driven along a straight line. The controller code will be written in C++ and a graphical user interface, GUI will be built using Python and a framework called Qt. The user should be able to plot signals and change the setpoint and controller parameters. The communication between the segway and the GUI should be done by wireless bluetooth. The state space model is already available. With that description, the project can be divided into four parts that the group should solve:

- Handling measurements
- Finding a working control law for balancing and setpoint following
- Implementing control code
- Implementing the graphical user interface

This report will include the resulting program structure, the operator communication and the control principles of the system. The resulting control parameters and a discussion about the control will also be included. The code for the GUI, the controller in C++ and the MatLab calculations can be found on GitHub. [4]

2 System Topology

The project task involves not only the stabilization of an inherently unstable process, but also the user interaction with a live control algorithm. That requires a solution for communicating between two instances: The controller executing on an Arduino and a PC that has to be able to hook onto the running control process. In the following the topology of those instances is outlined and discussed.

2.1 Initial Idea

The initial plan for the data structure can be seen in Figure 1 below. The idea was to use three threads: one for controlling the Arduino, one for the GUI, and one for the Bluetooth connection. The communication between the PC and the segway should be done with bluetooth.

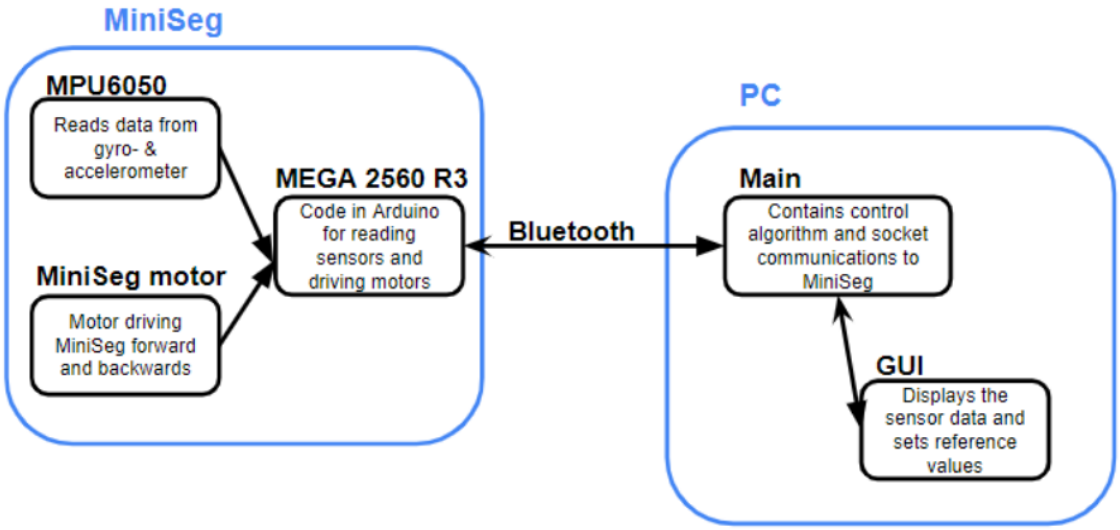

Figure 1: Illustration over the initial Data Structure

2.2 Development Approach

It was clear from the beginning that the development and implementation of a control algorithm for an inverted pendulum would be new terrain for every group member. Extensive testing and trial and error would be necessary to understand the sensors, controller calculations and the Arduino microcontroller in the first place. Therefore, it was quickly decided that a big portion of the project task would be to create a development environment that simplifies testing and minimizes the risk of manual programming errors.

The key challenge with this project was to enable the PC and the Arduino to communicate with each other robustly. So a platform independent data structure had to be found that could be used to transmit data between these instances. After some research it was found that the JSON format offers high flexibility and is widely used for communication in different

applications. Also, there are existing libraries for the Arduino as well as for Python that handle the task of serialization and deserialization of JSON data reliably. So it was decided to use this format for data transmission.

Furthermore, it had to be ensured that the communication was based on the same format of data. Therefore, a receive and transmit interface is defined centrally by specifying variable names and data types. Using this information, the Arduino and the GUI can use the Bluetooth connection consitently. The interface definition is the starting point for specifying exchangeable information and was altered multiple times during development.

A depiction of the final system topology can be seen in Figure 2 below. One of the biggest advantages of this data structure is the flexibility of the communication interface. One is able to change the behaviour of the controller dynamically since the communication link is two way. It is used to transmit information from the controller but also to send parameters set by the user. Therefore, the user is able to interface with the controller solely through the GUI which makes the process transparent, intuitive and user-friendly. The GUI thus takes the central role in this solution to the project challenge.

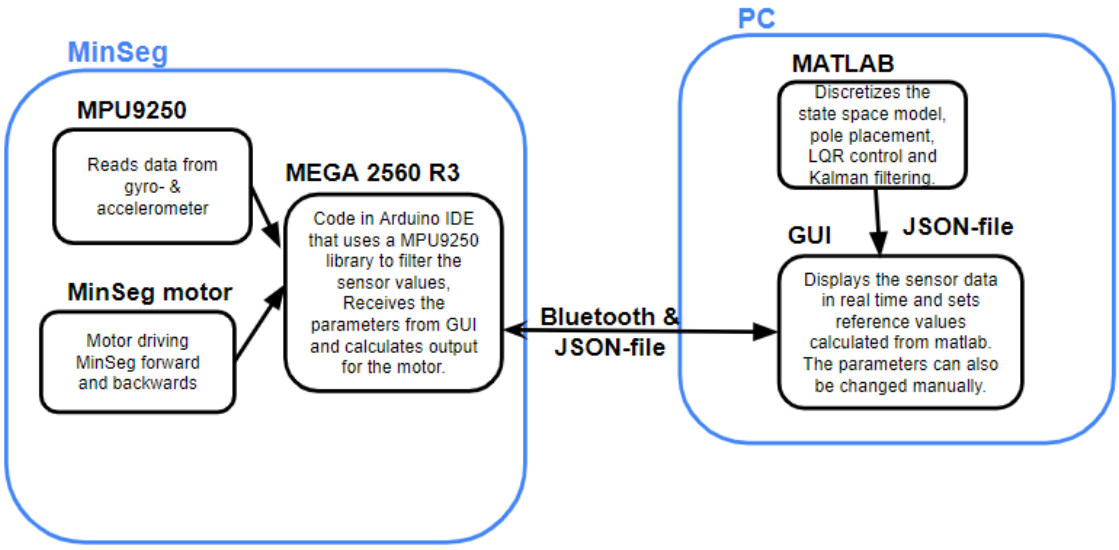

Figure 2: Illustration over the actual implemented data Structure

The actual controlling of the MinSeg is as previously mentioned written in C++ and executed on the Arduino. Here the readings of the data that the MPU9250 generates is done. This values are used in the code in order to make a correction of the state estimation. After the correction is made the calculation of the control signal according to the state estimation is made. After this the signal to the motor is sent, which is what controls it. Then the same values are used in order to put them in the prediction function of the observer implemented. Which is used to update the integral action state.

The MatLab code is used to calculate the control parameters and the observer parameters. This by using existing functions in MatLab for calculation for example the discretized the state space model, pole placement, LQR control and Kalman filtering. These parameters are saved in a JSON-file which allows for interfacing with the GUI.

The GUI is implemented in python and this is where the connection with Bluetooth is made. The calculated parameters from MatLab is accessed by loading in the JSON-file. This parameters can be sent with Bluetooth to the Arduino board. The GUI also displays the sensor data in real time and the parameters can also be changed manually.

3 Grafical User Interface

The graphical user interface enables the user to intuitively interact with the controller and monitor the internal variables in realtime. It is an important development tool since it provides a platform that can be used to quickly gather information on the control state as well as for testing different controller designs efficiently.

3.1 GUI development

In Figure 3 below an initial structure for the GUI is shown. The GUI were supposed to show the important parameters from the IMU (gyroscope and accelerometer) and the rotational encoder. These could then be translated to plot tilt angle to see how much the segway is tilting in either direction, the motors position using the encoder as well as the motors speed. Lastly the control input signal could be plotted. Parameters that could be changed by the user could be the sample time, the motors speed or the setpoint angle.

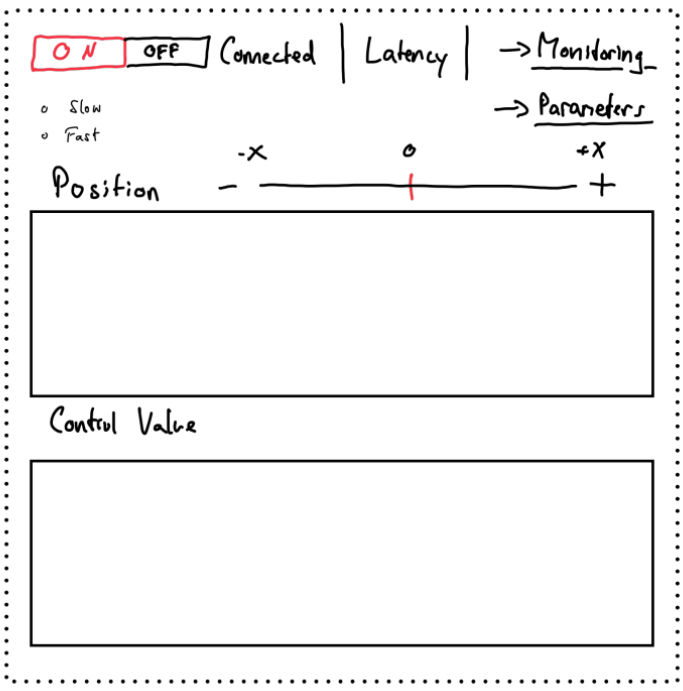

Figure 3: Possible screen layout for the GUI

The actual GUI can be seen in Figure 4 below. It includes a calibration mode to calibrate the initial position and tilt angle of the Segway. It is important that the segway is laid down on a flat surface when calibrating. The controller button should be on when the user wants to balance the Segway automatically. All the parameters can be changed and sent to the controller in real time while balancing the Segway. The readings from the IMU (gyroscope

and accelerometer) and the rotational encoder (i.e $\dot{\alpha}$, α, θ), as well as the actual output to the motor can be plotted if requested by the user.

Figure 4: Vizualisation of the final GUI

https://www.overleaf.com/project/644a80b3c5b19f7eecb87426

3.2 Functionality

The code and data structure is implemented to make the GUI as functional as possible. It has an easy and user friendly setup where basically all values from the sensors can be observed in different graphs. All parameters that are used by the Arduino can also be directly changed in the GUI. This makes it easy for the user to try different values.

As can be seen in Figure 4 a number of parameters can be changed manually in the GUI. There are four general values that can be changed, and these are the sample rate, an offset for the setpoint angle, a start and a stop offset for the motor actuation. The motor offsets change

the active range of the motor hence the relation between the control signal u and the 8 bit value that sets the PWM duty cycle. In reality the wheels don't rotate until a certain force is applied due to internal friction. The motor start offset lifts up the duty cycle value to a particular value that is chosen so that the motor is actually close to spinning when set. The motor stop offset is supposed to be lower than the motor start value and defines the control signal range where the motor is set to halt. Then there are three parameters related to the balance control that can be changed, and two parameters related to the position control.

Down in Figure 5 it can be seen exactly which different positions and angles can be shown through the GUI.

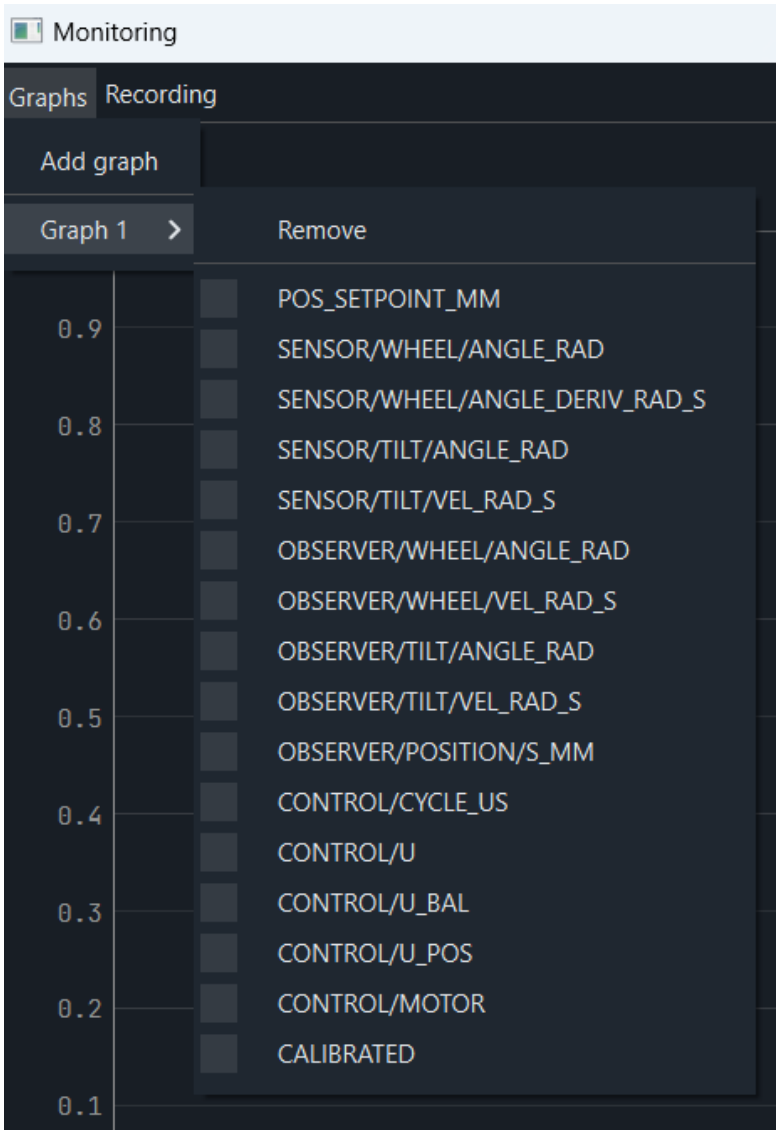

Figure 5: Showing what different values can be observed

The GUI enables the user to control the MinSeg to drive back and forth using the control panel, seen in Figure 4. The distance which the MinSeg should drive can also be set.

3.3 Workflow

Here follows a small and quick guide for what the GUI makes and how to use it in order to get the MinSeg to go back and forth and stabilize:

1. First of all make sure that the Arduino board already has been flashed with the latest code and then connect the segway to Bluetooth.
2. Once connected to Bluetooth a calibration of the segway should be done with the segway laying horizontal on a flat surface.
3. Make sure your Matlab values are saved in the JSON-file, and then choose "Parameters" and then "Load" to send them to the Arduino board.
4. There is also the possibility to change the parameters in the GUI manually. After the changes have been applied, "Confirm and Send" under "Parameters" has to be pressed to publish the information to the Arduino.
5. Then start the wheel actuation by turning the switch on.
6. Look and be proud over how the MinSeg is stabilized and moving back and forth.

4 Control Design

As a first design approach a conventional controller was implemented to verify the implementation and to evaluate the necessity of implementing a more advanced control theory. Therefore a simple state feedback design was implemented where the K-values of the state-representation were calculated by pole-placement. The following control law,

$$u(k) = -Kx(k) \tag{1}$$

was used for the state-feedback design. The K-values were calculated from the provided state-space representations of the system,

$$x(k+1) = \begin{pmatrix} -3.1 & 58.4 & 62.7 & 0 \\ 1 & 0 & 0 & 0 \\ 40.1 & -318 & -766 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} x(k) + \begin{pmatrix} -148 \\ 0 \\ 1808 \\ 0 \end{pmatrix} u(k) \tag{2}$$

$$y(k) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} x(k) \tag{3}$$

as well as the state-space vector, $x = [\dot{\alpha}, \alpha, \dot{\theta}, \theta\ ]^T$, where the values of the state-space vector were obtained from segway's IMU (gyroscope, accelerometer) in conjunction with the encoder of the wheels, i.e the following control variables could be measured directly:

- Tilt angle using the acceleration vectors, α
- Tilt velocity using the gyroscope, $\dot{\alpha}$
- The wheel angle using the wheel encoder, θ

The state space model shown above is given in a associated lab manual [1] and is referring to the reference system shown in Figure 6.

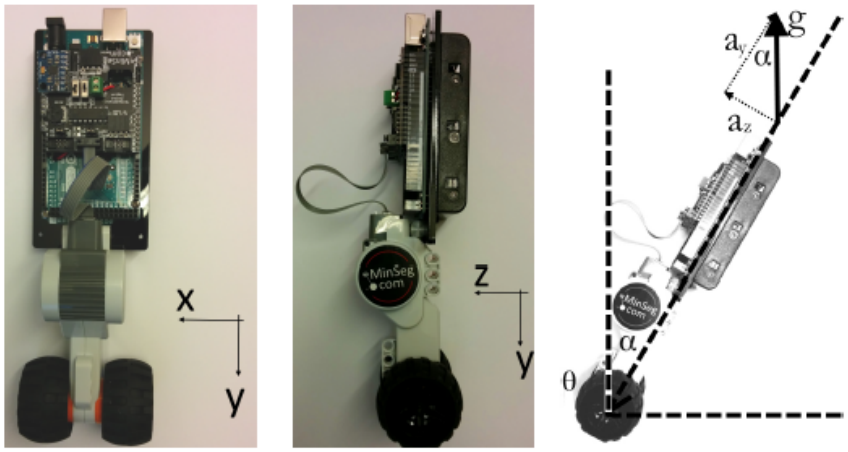

Figure 6: The robot coordinate system [1]. The tilted robot position shown is considered positive, so the rotation vector is collinear with the x axis consistent with a right hand coordinate system.

Different K-values were then tested iteratively. However since the segway was still unstable, a more advanced control design of the segway had to be implemented. To implement a more advanced control design the group utilized the mentioned lab manual [1], where a linear quadratic design approach was used together with a Kalman-filter.

4.1 LQR

Using Linear Quadratic Regulation one is able to find the optimal control parameters given the system model. To account for the requirement of position control, the control law was expressed as,

$$u(k) = -Kx(k) - k_i x_i(k) \tag{4}$$

this introduces a new state x_i which represents integral action on the position error the following way,

$$x_i(k+1) = x_i(k) + h(r(k) - x_4(k)) \tag{5}$$

where r is the wheel angle setpoint in radians. This extension results in a modified state space system model,

$$\Phi_{lqr} = \begin{pmatrix} \Phi_{11} & \Phi_{12} & \Phi_{13} & \Phi_{14} & 0 \\ \Phi_{21} & \Phi_{22} & \Phi_{23} & \Phi_{24} & 0 \\ \Phi_{31} & \Phi_{32} & \Phi_{33} & \Phi_{34} & 0 \\ \Phi_{41} & \Phi_{42} & \Phi_{43} & \Phi_{44} & 0 \\ 0 & 0 & 0 & -h & 1 \end{pmatrix} \qquad \Gamma_{lqr} = \begin{pmatrix} \Gamma \\ 0 \end{pmatrix} \qquad x_{lqr} = \begin{pmatrix} x \\ x_i \end{pmatrix} \tag{6}$$

that is used for the optimal control design. The optimal control parameters K and k_i are calculated in Matlab.

4.2 Kalman Filter

Early testing has shown, that the robot is very shaky and will rather destabilize itself when trying to run the control algorithm with the accelerometer, gyroscope and encoder sensor values. Especially the accelerometer is prone to disturbances since every tilt correction of the robot implies a force caused by the motor. Therefore, the tilt angle reading contains noise of high frequency. The used internal measurement unit MPU9250 from Invensense [2] has its own digital processing unit which applies a low pass filter on the accelerometer and gyroscope data. In this project, the filter bandwith of the accelerometer was configured to 45 Hz and the gyroscope filter bandwith to 20 Hz. However, the disturbances caused by the motor actuation are of lower frequency. A Kalman filter offers exactly that functionality to filter this type of disturbances that originate at the system input.

A simple system model is chosen to implement the state observer. The model for the tilt dynamics is depicted in Figure 7 and for the wheel dynamics in Figure 8.

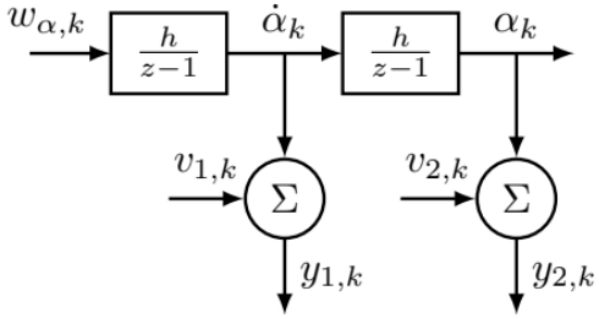

Figure 7: The observer model for the tilt dynamics [1].

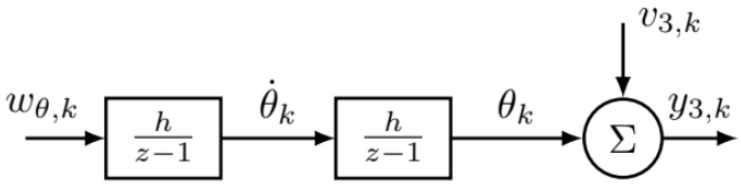

Figure 8: The observer model for the wheel dynamics [1].

These discrete models are simplified to double integrators and to take inputs w_α and w_θ that represent the disturbed input signals, so the disturbance itself is not separated from the

actual system input. Despite this simplification, the model is able to approximate the system dynamics sufficiently. They translate to the state space system described the following way,

$$x_{obs}(k+1) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ h & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & h & 1 \end{pmatrix} x_{obs}(k) + \begin{pmatrix} h & 0 \\ 0 & 0 \\ 0 & h \\ 0 & 0 \end{pmatrix} w(k) \qquad w = \begin{pmatrix} w_\alpha \\ w_\theta \end{pmatrix} \qquad x_{obs} = x \tag{7}$$

where the system input is denoted as w. The estimated state is calculated by a observer with direct term. Where the following equation,

$$\hat{x}(k+1|k) = (\Phi_{obs} - LC)\hat{x}(k|k) + Ly(k) \tag{8}$$

refers to the state's prediction term, followed by its correction term shown here,

$$\hat{x}(k|k) = \hat{x}(k|k-1) + M_x(y(k) - C\hat{x}(k|k-1)) \tag{9}$$

The observer's implementation is closely guided by the documentation for Kalman filtering by MathWorks [3]. The notation $x(k_1|k_2)$ is used to additionally hint that the value $x(k_1)$ is calculated at cycle k_2. Using this notation, it can be deduced that the state vector is calculated by equation (8) at a control cycle earlier than when it is used for calculating the control signal. The correction, equation (9) is performed just before the control signal is calculated using the result of the state prediction. An advantage of the observer with direct term is that all measurements y up to cycle k are considered. An observer without direct term lacks the correction term thus it only considers measurements until cycle $k-1$.

4.3 Feedforward Control

Using only integral action for position control and define the control law as in Equation 5 has a major drawback: The system reacts very slowly and tends to overshoot or become unstable if the impact of integral action is increased. However, it is still indispensible for stationary accuracy since it accounts for unexpected disturbances and model inaccuracies. Using Feedforward Control one is able to combine stationary accuracy as well as quick system response upon a setpoint change.

In "Wittenmark et al. [5]" a feedforward design for state feedback controllers was introduced. It features a system model to incorporate trajectory control upon a setpoint change. This

means that the control error is introduces so that the system will move in a certain way which is compatible to the system dynamics when the setpoint changes. The control law is defined by,

$$u(k) = K(x_m(k) - x(k)) + u_{ff}(k) \tag{10}$$

meaning that the control signal consists of a feedback and a feedforward part. The model state x_m is calculated by a state space system similar to the given state-space (see equation (2)),

$$x_m(k) = \Phi x_m(k) + \Gamma u_{ff}(k) \tag{11}$$

where the matrices Φ and Γ are chosen identically to equation (11), only the system input is defined differently. The input of the model state space is defined to be the feedforward control signal defined by,

$$u_{ff}(k) = -K_m x_m(k) + K_c u_c(k) \tag{12}$$

using this definition, the dynamics of the model can be modified by choosing K_m appropriately. For this purpose, one can also use linear quadratic regulation and find the optimal value for K_m. One can define the cost function for this optimization differently than the one defined when calculating the optimal controller parameter K in order to achieve a certain setpoint response. K_c is calculated by,

$$K_c = \frac{1}{C(I - \Phi + \Gamma K_m)^{-1}\Gamma} \qquad C = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix} \tag{13}$$

Equation 13 to obtain a static gain of 1 in respect to u_c being the wheel angle setpoint.

4.4 Controller Structure

Figure 9 depicts an overview of the implemented controller. A similar model was implemented in Matlab Simulink and used for validating the parameters and the control approach.

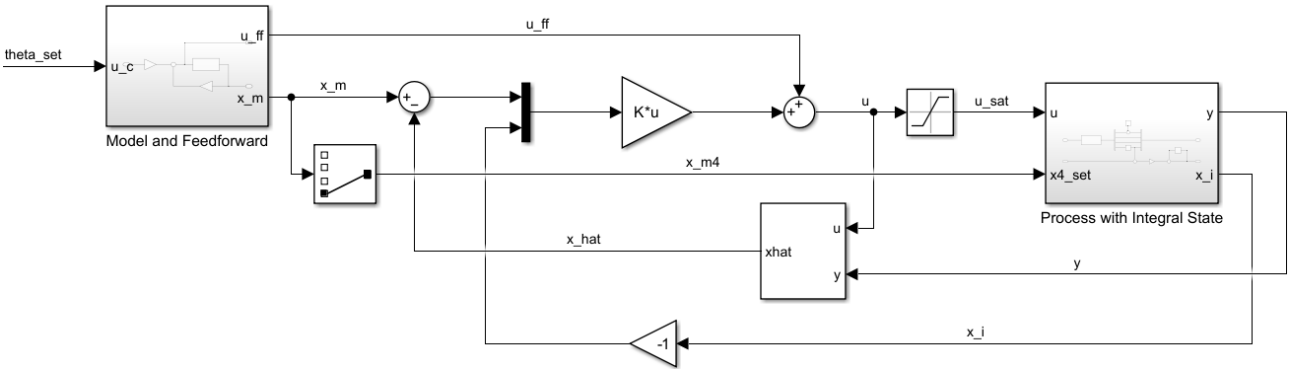

Figure 9: Block diagram of the implemented controller.

In summary, the controller features three main steps:

1. Read the measurements and estimate the current system state.
2. Generate the feedforward control signal and model states based on the setpoint provided.
3. Calculate the feedback control signal and set the motor voltage.

It is worth noting that after introducing the feedforward approach, the wheel angle setpoint r that is used for calculating the integral action state according to Equation 5 is not the setpoint directly given by the user. Instead, the model state $x_{m,4}$ is used for that to make the integral action compatible with the feedforward approach.

5 Result and discussion

The control gain, the kalman filter and the feedforward model were designed using respective optimization matrices considering a sample rate of $h = 6ms$. The sample rate is limited to its lower end by the low pass filter delay of the IMU. h must be configured so that the value is higher than the delays, otherwise calculations would use the same measurements repeatedly. The following,

$$Q_{lqr} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1e4 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad R_{lqr} = 1 \tag{14}$$

shows the parameters that lead to a controller which put more weight on the correction of the robot tilt angle. The Kalman filter was supposed to smooth out the tilt angle measurement, which is why it puts most of the weight on the respective disturbed signal. As can be seen below:

$$Q_{kalman} = \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix} \qquad R_{kalman} = \begin{pmatrix} 1e-9 & 0 & 0 \\ 0 & 1e-9 & 0 \\ 0 & 0 & 1e-9 \end{pmatrix} \tag{15}$$

The feedforward model was designed using the matrices below:

$$Q_{ff} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad R_{ff} = 10 \tag{16}$$

This results in a trajectory that converges slowly but securely to the given position setpoint since the output signal dynamic is penalized.

These design choices enable the MinSeg to balance and respond to position setpoints successfully. However, there are limitations regarding big setpoint changes. This is due to inaccuracies in the system model which lead in combination with the feedforward approach to control signal that are too ambitious for motor to handle, ultimately causing the robot to fall. The most important results are discussed in the following chapters.

Below in Table 1 a customized control gain vector is shown. These values were determined iteratively to account for the inaccuracies of the system model. The robot seems more secure and calm when using these parameters. However, the setpoint step size which is controlable is still limited.

Table 1: The best values found for K

Parameter	**Value**
K_1	-15.810
K_2	-106.052
K_3	-1.018
K_4	-0.720
K_i	0.300

5.1 Balancing Task

The result when balancing the MinSeg and when using both LQR control an Kalman filter can be seen in Figure 10 below. Graph 1 shows the derivative value of the encoder, $\dot{\theta}$ both the actual and the observer value. As the graph can tell, the filtered value is the same as the value received from the sensor. Graph 2 shows the wheel angle, θ, for the sensor and the observer, and as before the filter value is the same as the sensor. The observer is not relevant for the encoder values.

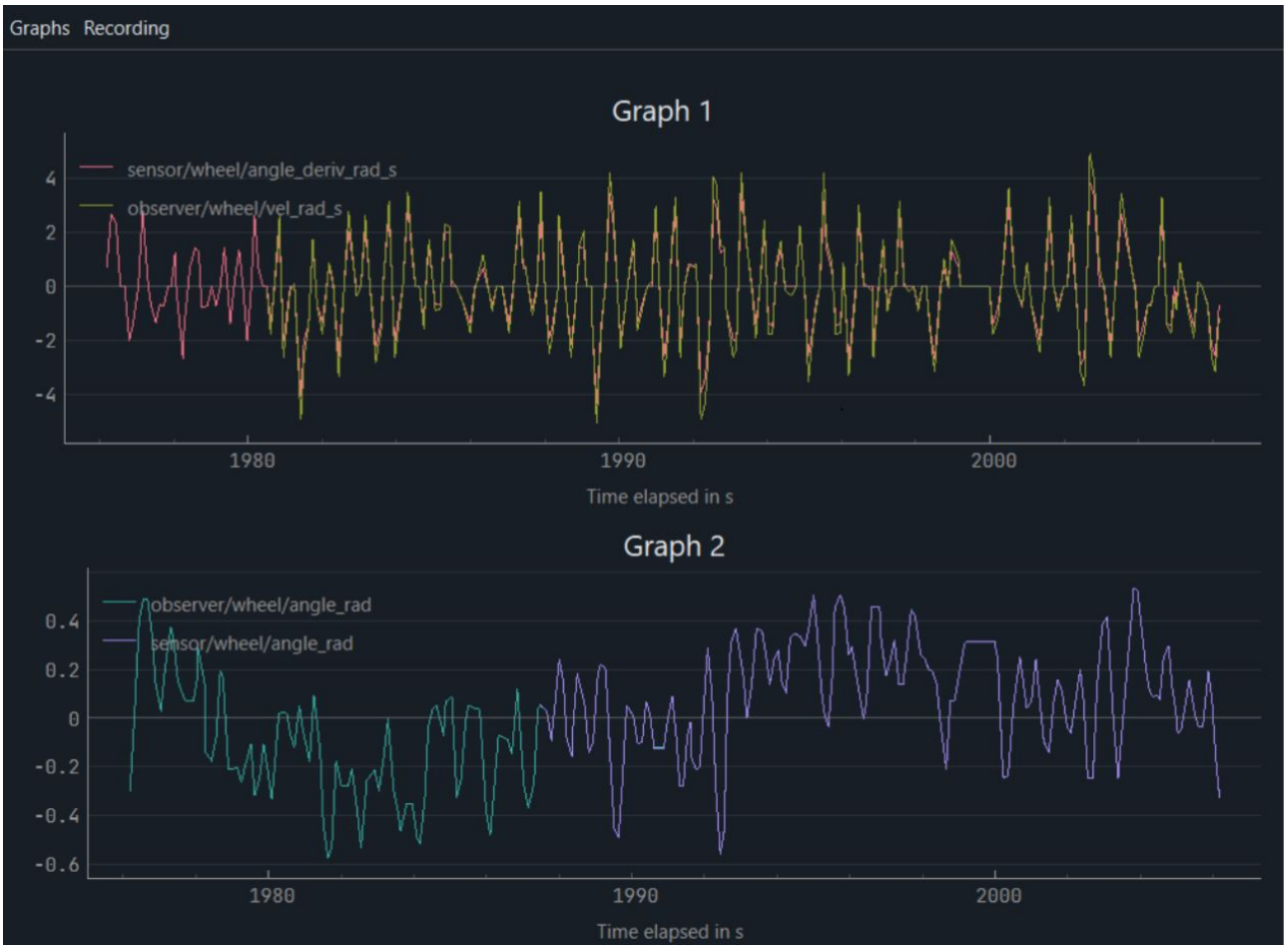

Figure 10: Comparison between sensor and observer value for $\dot{\theta}$ and θ

The kalman filter is however more relevant when handling the tilt angle, α values as can be seen in Figure 11. Even though the value from the sensor is sent through a low pass filter when using the MPU9250 library it is still volatile and if used directly this will cause problems in balancing. The observer, in the graph visualized in blue, is more even which will result in a more stable system. It was not until the implementation of the Kalman filter that the MinSeg was able to stand for a longer time and be controlled properly.

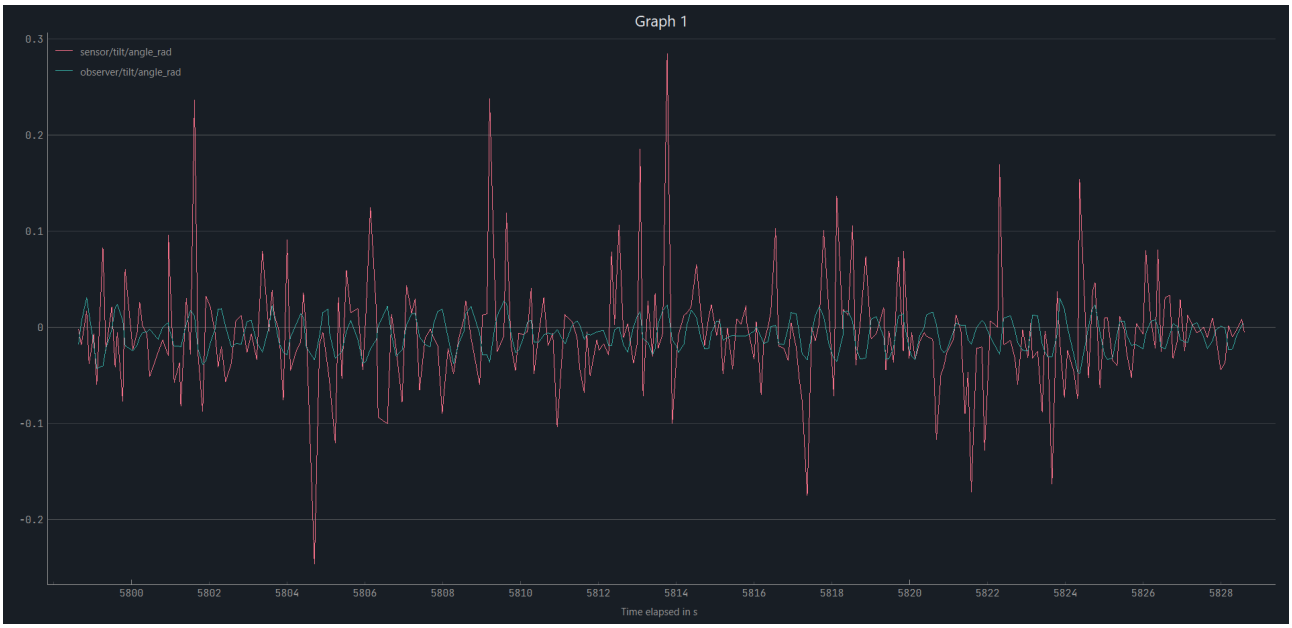

Figure 11: Comparison between sensor and observer value for α

Figure 12 shows the position and tilt angle values compared to the setpoint when balancing and using the more stable customized controller gain parameters from Table 1. The robot experiences a major disturbance at around 30 seconds but manages to stabilize itself again. Additionally, the robot tries to control the tilt angle to a value that is slightly different to 0. This offset was determined empirically and is the real balancing point. If the weight distribution of the MinSeg would be ideal, this offset would be 0.

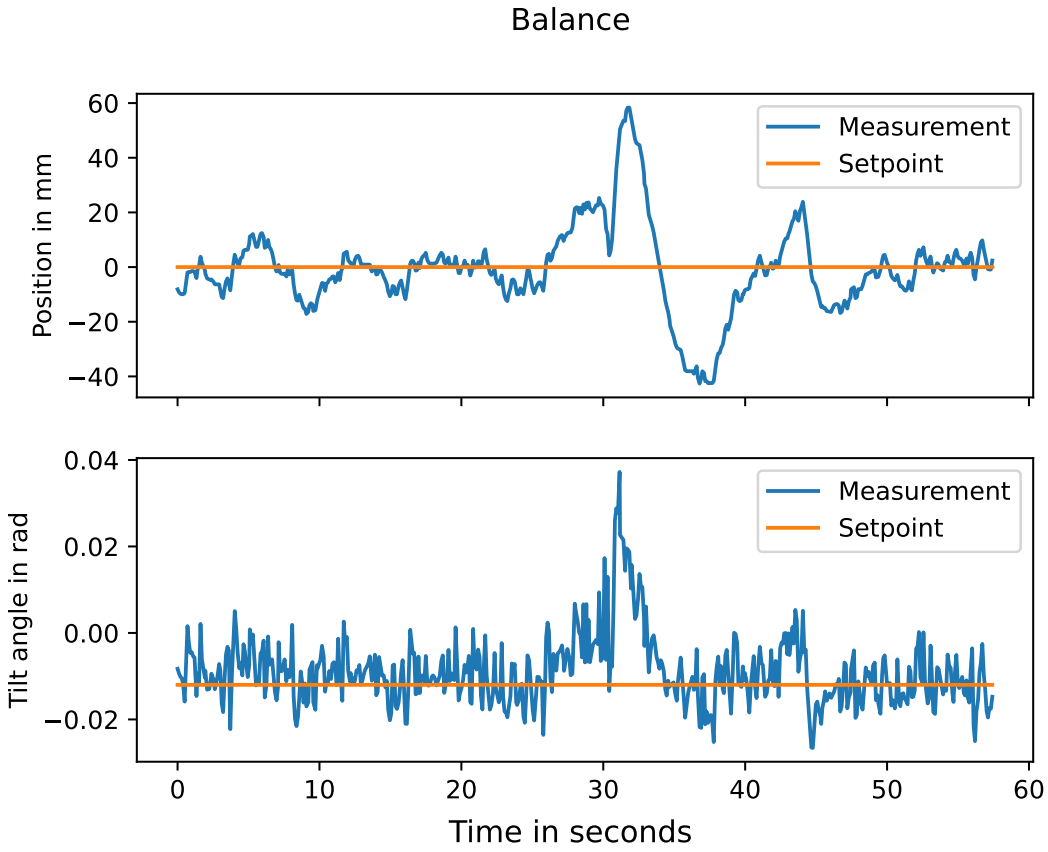

Figure 12: Balance task result

5.2 Position Control Task

In Figure 13 the system response to a setpoint step is depicted. Again, using the custimzed control gain from Table 1. Using the feedforward model, the system tries to follow the given trajectory instead of the setpoint step directly. Ideally, this should result in a consistent movement. However, due to the model inaccuracies, the system struggles to follow the model trajectory. When the setpoint step is induced, one can observe that the movement is initiated successfully. Just after that, the system overtakes the model trajectory and the controller has to correct the robot's position accordingly. This results in an inconsistent movement throughout the approach to the steady state. To make the best use out of the feedforward approach, the system model needs to be improved to better represent the actual system. Nevertheless, the robot manages to respond to setpoint changes successfully.

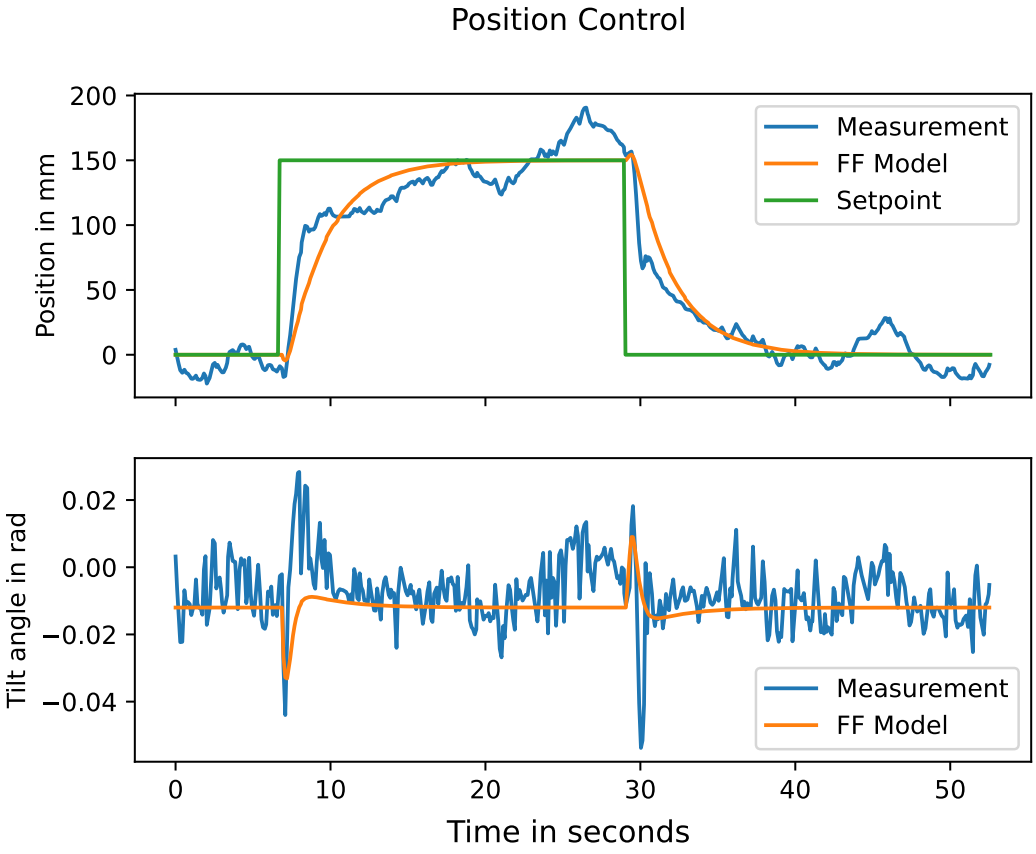

Figure 13: Position control task result

6 Conclusion

The most difficult problem during this project was to understand which direction to choose when implementing a working control law. In the beginning, the group aimed for making the MinSeg balance by only pole placement and discretizing the continuous system, but trying to fix the parameters without knowing where to place the poles was not optimal. The group did not have sufficient knowledge about pole placement, and it might not be possible to make this unstable system stable without implementing a more advanced control. Since the GUI was already working at this point, it was easy to change the parameters and try different values, but the result did not end up in a balanced MinSeg. It was not until the Kalman filter and LQR control were implemented that the SegWay could balance. This result is reasonable when looking at Figure 11, where the difference in the filtered signal and the non-filtered signal is very noticable. In conclusion, it is a good approach to use Kalman filter and LQR control when the values of sensors are volatile, therefore making it difficult to accurately place the poles of the system.

Overall, this project surely made every project member learn a lot about applying the theory of state space controlling to a complex control task. It helped to make the abstract concepts of designing a multi input controller graspable. In the end, the MinSeg handled the challenge of balancing and setpoint following reliably.

7 Literature

[1] Anton Cervin and Nils Vreman. *Kalman Filtering and LQ Control of the MinSeg Robot*. Lund University, 2022.

[2] InvenSense Inc. *MPU-9250 Product Specification Revision 1.1*. 2016.

[3] Inc. MathWorks. *Kalman Filtering*. URL: https://de.mathworks.com/help/control/ug/kalman-filtering.html. Accessed: 11.05.2023.

[4] Robin Müller. *Minseg*. 2023. URL: https://github.com/robin-mueller/minseg.

[5] Björn Wittenmark, Karl Johan Åström, and Karl-Erik Årzen. *Computer Control: An Overview Educational Version 2021*. 2021.