SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

Faculty of Electrical Engineering and Information Technology

Registration number: FEI-104376-106226

Control System Synthesis

Diploma Project 1

Study Programme:	Robotics and Cybernetics
Study Field:	Cybernetics
Training Workplace:	Institute of Robotics and Cybernetics
Supervisor:	Ing. Marián Tárník, PhD.

2026 **Bc. Radovan Jakubčík**

Slovak University of Technology in Bratislava
Institute of Robotics and Cybernetics

Faculty of Electrical Engineering and Information Technology
Academic year: 2025/2026
Reg. No.: FEI-104376-106226

MASTER THESIS TOPIC

Student:	**Bc. Radovan Jakubčík**
Student's ID:	106226
Study programme:	Robotics and Cybernetics
Study field:	Cybernetics
Thesis supervisor:	Ing. Marián Tárník, PhD.
Head of department:	prof. Ing. František Duchoň, PhD.
Workplace:	Institute of Robotics and Cybernetics

Topic: **Control System Synthesis**

Language of thesis: English

Specification of Assignment:

The objective of this diploma thesis is to design a comprehensive control system for a selected laboratory device representing a physical model of a dynamic system. The work will employ advanced methods of automatic control, with a particular focus on robust and adaptive control techniques.

Tasks:
1. Provide a detailed description of the selected technical device or process from the perspective of systems modeling and control. Define the input and output variables, characterize its static and dynamic properties, and assess the possibilities for its mathematical modeling.
2. Review and describe the control methods and algorithms that will be applied in the design of the control system.
3. Using simulation and analytical tools, demonstrate sample control results, including a relevant evaluation of control performance.
4. Address the practical aspects of implementing the proposed control algorithm using embedded microcontroller systems or programmable logic controllers (PLCs).
5. Evaluate the achieved results and prepare a written thesis documenting the solution of the assigned tasks.

Deadline for submission of Master thesis:	15. 05. 2026
Approval of assignment of Master thesis:	06. 11. 2025
Assignment of Master thesis approved by:	prof. Ing. Jarmila Pavlovičová, PhD. – Study programme supervisor

Contents

List of Symbols and Abbreviations

μs	Microsecond
AS	AeroShield
DC	Direct Current
EKF	Extended Kalman Filter
GPIO	General-Purpose Input/Output
I2C	Inter-Integrated Circuit
KF	Kalman Filter
LQR	Linear Quadratic Regulator
LTI	Linear Time Invariant
MCU	Microcontroller Unit
MPC	Model Predictive Control
ms	Millisecond
MSE	Mean Squared Error
NMPC	Nonlinear Model Predictive Control
NSS	Nonlinear State-Space
OP	Operating Point
PWM	Pulse-Width Modulation
SCL	Serial Clock Line
SDA	Serial Data Line
SS	State Space
USART	Universal A/synchronous Receiver Transmitter

Introduction

World around us is ever changing and dynamic. To effectively synthesize and analyze such systems, we resort to mathematical modeling. Mathematical models help us understand the behavior of systems, predict their responses to various inputs, and design control strategies to achieve desired outcomes.

Within this thesis we will be focusing on a single system, analyze it from the perspective of identification and control design. Identifying the system's inputs, outputs, dynamics and static characteristics. To better illustrate the process hidden behind simple terminology of system identification and control design. We will attempt to cover the entire process from the ground up, starting from system analysis, model derivation, identification, controller design, simulation, implementation and finally comparison of the different approaches used throughout the thesis. The goal is to provide a comprehensive guide of the entire process of system identification, controller design, and validation, such that the reader can apply these concepts to other systems in the future. The methods used within the thesis will mainly focus on non-linear system identification and non-linear control design, as these methods are more general and can be applied to a wider range of systems, without the limitation of linearization around a certain operating point (OP). Experiments will be proposed and conducted to gather necessary data for the identification process. Designing correct experiments is crucial to ensure that the data collected represents the systems behavior accurately. Furthermore, the process of deriving a physical model of the system can be done without having the collected data. However, having experimental data allows for validation of the derived model and ensures that it accurately represents the real-world system.

With this in mind, we will provide a simple yet effective methodology for deriving a non-linear model of a pendulum type system using Lagrangian mechanics, which can further be used in various similar systems; for example robotic arms, inverted pendulums, and other mechanical systems with rotational dynamics. We will focus on deriving a non-linear state-space (NSS) model, show how to linearize it around a certain OP, giving us the linearized state-space (SS) model. Discretization of both models will be covered. Which comes in handy when designing discrete controllers to be implemented on microcontroller units (MCU). We will design various discrete controllers, including but not limited to linear quadratic regulator (LQR) and model predictive control (MPC). Simulations will be conducted to validate the derived models and designed controllers. Finally, the designed controllers will be implemented on an MCU, and their performance will be compared to the simulation results, to demonstrate the difference between theoretical and practical applications.

1 System Analysis

Within this section, we will analyze the system in detail, covering its description, inputs, outputs, disturbances, and both static, dynamic characteristic, and impulse response. For each of the aforementioned characteristics, we will provide experimental results obtained from the real system and the process of gathering the data. Thus, proposing suitable experiments for measuring the respective characteristics. Implementing the experiments on the real system with the help of an MCU and MATLAB scripts for data acquisition and processing.

With the understanding gained from this analysis, we will discuss the possibility of deriving a physical model of the system. Based on the conclusion of this discussion, we will either proceed to develop the physical model or identify a linearized SS representation of the system at a certain OP for further controller design.

1.1 System Description

The system under consideration is an actuated (propelled) pendulum consisting of a mass attached to the end of a rigid rod, which is free to swing in a vertical plane under the influence of gravity as can be seen on fig. 1. The pendulum is actuated by a propeller directly attached to the motor shaft of a separately excited direct current (DC) motor, which represents the attached mass on the end of the rod.

The motor is powered by a DC voltage source. Allowing for control over the pendulum's motion by controlling the pulse-width modulated (PWM) signal applied to the gate of a transistor from the microcontroller. This transistor acts as a switch, regulating the voltage supplied to the motor based on the duty cycle of the PWM signal. The MCU cannot directly control the voltage supplied to the motor, as it can only output digital signals (high or low voltage levels), which in logic level correspond to either $5V$ or $0V$ in our case, when using the Arduino Uno R3.

Table 2 Ranges and units of signals

Signal	Value Range	Unit
Input	0% to 100%	[%]
Output	$-50°$ to $210°$	[°]
Potentiometer	0% to 100%	[%]

This means that the control input to the system is the duty cycle of the PWM signal, which can vary from 0% to 100% - $0V$ to $5V$ respectively, to open or close the transistor. The control input signal is provided by the microcontroller unit (MCU)'s pin $\sim D5$. The

pendulum's angular position is measured using a rotary magnetic encoder AS5600 mounted at the pivot point of the pendulum, which provides high-resolution angular position feedback within the range $-50°$ to $210°$. At last an integrated potentiometer is included in the system, which can be used for various purposes, such as early termination of a simulation when the signal value exceeds 90%, or as a reference input for the controller. The potentiometer signal ranges from 0% to 100%. All the signal ranges and their respective units can be found summarized in tab. 2.

Figure 1 The considered actuated pendulum system named AeroShield (AS) device.

The pendulum itself has a physical angular constraint from $-50°$ to $210°$, which is enforced by two physical stoppers located at these angles. We will not be considering these constraints in our modeling and controller design, as we will be operating the

system within these limits. Although we will attempt to account for these constraints during simulation. The propeller attached to the motor shaft generates thrust, which produces a torque around the pivot point of the pendulum, allowing us to control its angular position. This torque generated by the propeller only acts in one direction, meaning that it can only accelerate the pendulum in the positive angular direction. Thus, to decelerate or move the pendulum in the negative angular direction, we will rely on gravitational forces and frictional forces acting on the system.

Whenever the pendulum is in negative angular position, it is either the consequence of initial conditions (e.g., starting from rest at a negative angle; manually moved to a negative angle) or due to gravitational forces pulling it downwards when the propeller is not generating enough thrust to counteract gravity.

Let us make prior assumptions about the system: if the pendulum is to be held at a 90° angle, the propeller must generate the highest amount of thrust to counteract the gravitational torque acting on the pendulum. When the pendulum is above 90°, the propeller must generate less thrust, as gravity will assist in holding the pendulum in place. The process of gravity assisting the propeller in holding the pendulum in place will remain true until the pendulum reaches the angle of 180°, after which gravity will start to oppose the propeller's thrust again, requiring more thrust in reverse direction to hold the pendulum in place. This implies the system is controllable within the angular range of 0° to 180°.

1.2 Communication

Throughout the thesis, we will be utilizing serial communication USART between the MCU and MATLAB for data acquisition and control signal transmission. The MCU will be responsible for reading the angular position from the rotary encoder, generating the PWM signal to control the motor, and sending the measured data to MATLAB for further processing and analysis. MATLAB will handle the the more demanding tasks, such as data logging, real-time visualization, and controller computations, leveraging powerful computational capabilities on the host computer. This is especially useful during the experimental phase, where rapid prototyping and testing of different control strategies are required. This approach limits us to timing constraints imposed by the serial communication speed and MATLAB's processing time. Therefore, we will mainly choose sampling times in the range of 20ms to 100ms - where *ms* stands for milliseconds, see the tab. 4 for details.

The synchronization between the MCU and MATLAB will be handled by MATLAB, which will send commands to the MCU in the form of control input, to which the MCU will respond by sending back all the defined data, including the measured angular

position and any other relevant signals. To ensure optimal performance, we will be using a baud rate of 250 000 for the serial communication. The data is stored within a union object containing the structure of the required data and its byte representation on the MCU side. Allowing us to send and receive the data in binary format, minimizing the amount of data transmitted over the serial link and reducing latency. The data structure used for sending the data from the MCU to MATLAB is defined in listing 1 and more details can be found in tab. 4.

Listing 1: Data structure used for serial communication between the MCU and MATLAB.

```
struct __attribute__((packed)) AeroData
{
    unsigned long time;
    float output;
    float control;
    float potentiometer;
    unsigned long control_time;
    unsigned long dt;
};
union AeroDataUnion
{
    AeroData data;
    byte bytes[sizeof(AeroData)];
};
```

Table 3 Communication configurations

Configurations	Value/Range	Unit
Baud rate	250 000	[bps]
Sampling time	20 to 100	[ms]
Out bytes	24	[bytes]
In bytes	4	[bytes]

Table 4 Data structure configurations

Configurations	Value/Range	Unit	Byte size	Type
time	0 to 4 294 967 295	[μs]	4	unsigned long
output	-50 to 210	[°]	4	float
control	0 to 100	[%]	4	float
potentiometer	0 to 100	[%]	4	float
control_time	0 to 4 294 967 295	[μs]	4	unsigned long
dt	0 to 4 294 967 295	[μs]	4	unsigned long
data packet size			24	bytes

MATLAB unpacks the received byte array back into the defined structure within a custom class object for easier access to the individual data fields and increased

user-friendliness. While MATLAB sends a single floating point value being the control input to the MCU. An example of the communication process can be seen in fig. 2.

1.2.1 Communication Timing Measurement

The process behind measuring the communication timing was to design an experiment for realizing the measurement. The proposed experiment involved attaching PicoScope 4000 series high resolution oscilloscope probes to the MCU's general-purpose input/output (GPIO) TX and RX pins.

Afterwards, we need to arrange for an ongoing communication between MATLAB and the MCU, thus we run a simple script in MATLAB to send a constant control input of 20% to the MCU at a sampling time of 20ms. Now we have to capture the start of the communication cycle. Firstly, we have to define what the start of the communication cycle is. Let us define it as the event when MATLAB starts sending the control input to the MCU. Thus, to capture this event, we set the oscilloscope to trigger on a rising edge on the RX pin of the MCU. After setting up the trigger, we can proceed to capture the communication cycle.

We set the oscilloscope to capture a 10ms time window, which is sufficient to capture the entire communication cycle. We repeat this process 20 times to ensure accuracy and consistency of the measurement. The captured data is then analyzed to measure the time taken for the entire communication cycle, from the moment MATLAB sends the control input to the moment the MCU ends sending the data back to MATLAB. We measure the time difference between the rising edge on the RX pin and the end of the transmission on the TX pin, which gives us the total time taken for the communication cycle.

Except for monitoring the communication timing, we also observer the order in which the data is sent and received, ensuring that the data integrity is maintained throughout the communication process. While keeping the planned order of operations intact. Where it starts with MATLAB sending the control input, followed by the MCU processing the input, reading the sensor data, and finally sending the measured data back to MATLAB. Without having the order of operations verified, we cannot be certain that whatever control strategy we implement, will function as intended, since the data being used for control computations might be outdated or incorrect. This entire process is illustrated in fig. 2.

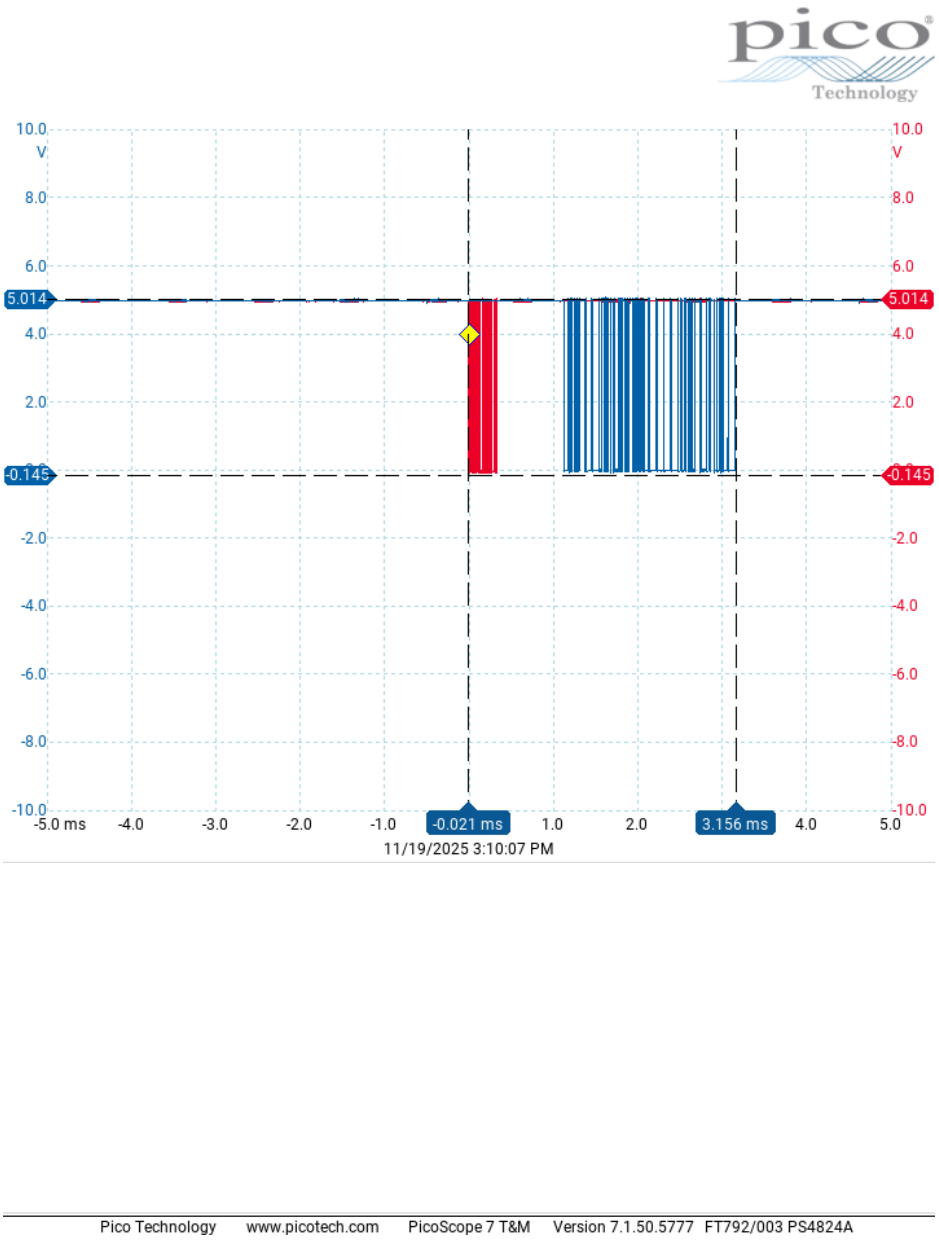

Figure 2 Communication process between the MCU and MATLAB, showcasing the proposed order of operations.

Where the red colored signal represents the control input from MATLAB and the blue colored signal is for the data sent to MATLAB. The average time taken for the communication cycle was found to be approximately 3.156ms.

We have to assume the MCU might skip a read operation, if the data arrives right after the MCU attemptes to read it from the RX buffer, thus skipping the entire

read-process for that one cycle. Resulting in the brief pause in the communication, before sending the data back to MATLAB. Or another likewise scenario is when the MCU read the data, but the time it took to process the measurement, was long enough to result in the blank spot in the communication. This example was measured while using a baud rate of 115 200, meaning with baud rate set to 250 000 the time taken for the retrieval and transmittion of data is even lower.

1.2.2 System's schematic diagram

The schematic diagram of the system is shown in fig. 3, where the measurable and controllable signals are illustrated along with the potentiometer signal.

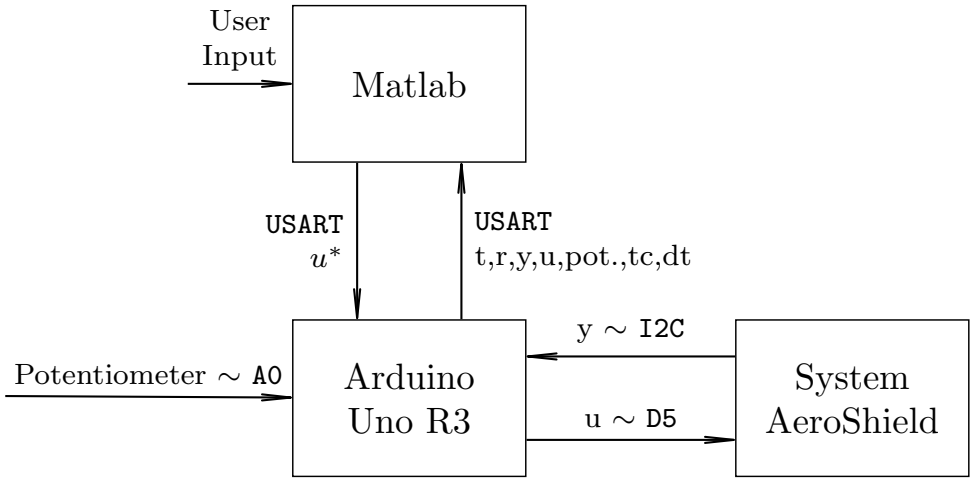

Figure 3 Schematic diagram of the communication protocol implemented for the AS system.

We can see the scheme fig. 3 illustrates the communication without explicitly specifying the order of operations, which was measured and verified in the previous section sec. 1.2.1. This schematic diagram servers as a high-level overview of the system's communication architecture. Where all the used communication channels are portrayed, while the order is not explicit, it can be inferred from the fact that the scheme contains input from the user, which represents the start of the measurement in MATLAB.

The transmitted optimal control input in diagram abbreviated as u^* to the MCU, is the requested control input for the current time step $u^*(k)$, either computed by the control law or set to a constant value for open-loop measurements.

1.3 Control Input

While our system is controlled from MATLAB, the control input is used as the synchronization signal for the entire system. The control input received from MATLAB is further processed on the MCU side to generate the appropriate PWM signal for controlling the motor, which in turn controls the pendulum's motion. The control input signal is defined as a percentage value ranging from 0% to 100%, representing the duty cycle of the PWM signal. Yet, this percentage value is further mapped onto a single byte value ranging from 0 to 255, which is the actual value used to set the duty cycle of the PWM signal on the MCU. This mapping is done using a simple linear transformation, where 0% corresponds to a byte value of 0, and 100% corresponds to a byte value of 255. The mapping function can be expressed mathematically in the following form

$$\mathrm{PWM}(u) = \left(\frac{u}{100}\right) \times 255 \tag{1}$$

where PWM is the byte value used to set the duty cycle of the PWM signal, and u is the control input percentage value received from MATLAB. This costs us a small loss in resolution, as we are mapping a continuous range of values onto a discrete set of 256 possible values. To be safe the (1) input is saturated within the range of 0% to 100% before applying the mapping. Let us define the saturation function as follows

$$\mathrm{sat}(u, u_{\min}, u_{\max}) = \begin{cases} u_{\min} & \text{if } u < u_{\min} \\ u & \text{if } u_{\min} \leq u \leq u_{\max} \\ u_{\max} & \text{if } u > u_{\max} \end{cases} \tag{2}$$

where sat is the saturation function, u is the input value to be saturated, $u_{\min}$ is the minimum allowable value, and $u_{\max}$ is the maximum allowable value. The other approach to defining the saturation function is by using the `min` and `max` functions as follows

$$\mathrm{sat}(u, u_{\min}, u_{\max}) = \min\{\max\{u, u_{\min}\}, u_{\max}\} \tag{3}$$

where the variables have the same meaning as in (2) and have to obey the following condition $u, u_{\min}, u_{\max} \in \mathbb{R}$ and $u_{\min} < u_{\max}$. The $\{\}$ brackets denote the set notation, which is used here to indicate the arguments of the `min` and `max` functions, simply said we are choosing minimum and maximum out of a set of values. Using either of the two definitions of the saturation function will yield the same result. Yet, it uses different mathematical operations to achieve the same outcome and can be chosen based on personal preference or specific requirements of the implementation. Or because functions like `min` and `max` are often optimized in programming languages and libraries, making them potentially more efficient for certain applications.

Reading the control input from MATLAB in the MCU is done using the serial communication interface, where the MCU continuously checks for incoming data on its RX buffer. When new data is available, more precisely when the number of bytes available in the RX buffer is equal to the size of a single floating point value (4 bytes), the MCU reads the data from the buffer and stores it in the data structure defined in listing 1. This ensures that the control input is always up-to-date and ready to be used for generating the PWM signal. The same control input which was read from MATLAB is also sent back to MATLAB as part of the data structure, because it specifically belongs into the very step of the control loop - ensuring no discrepancies arise.

1.4 Measurable Output

The measurable output of the system is the angular position of the pendulum, which is measured using a rotary magnetic encoder AS5600. The encoder provides high-resolution angular position feedback via the Inter-Integrated Circuit (I2C) communication protocol. Because of the predefined resolution of the encoder, the time it takes to read the angular position is non-negligible, thus we have to account for this delay in our system analysis and controller design.

We can measure this delay experimentally by using the same principle as in the communication timing measurement experiment described in the sec. 1.2.1. We set up the oscilloscope to trigger on the rising edge of the control input signal sent from MATLAB to the MCU. Then we measure the time difference between this rising edge and the moment when the MCU starts reading the RX buffer, because we assume that the reading of the angular position only happens after the control input is received, because that is the order of operations we defined earlier. See fig. 5 for the order operations defined and measured in real-time.

Yet, now we have to attach other probes to monitor the I2C clock (SCL) and data lines (SDA), to determine the exact moment when the angular position reading starts. After conducting the experiment, we found that the average time taken for reading the angular position from the encoder is approximately 546.3μs. This measurement can be seen in fig. 4.

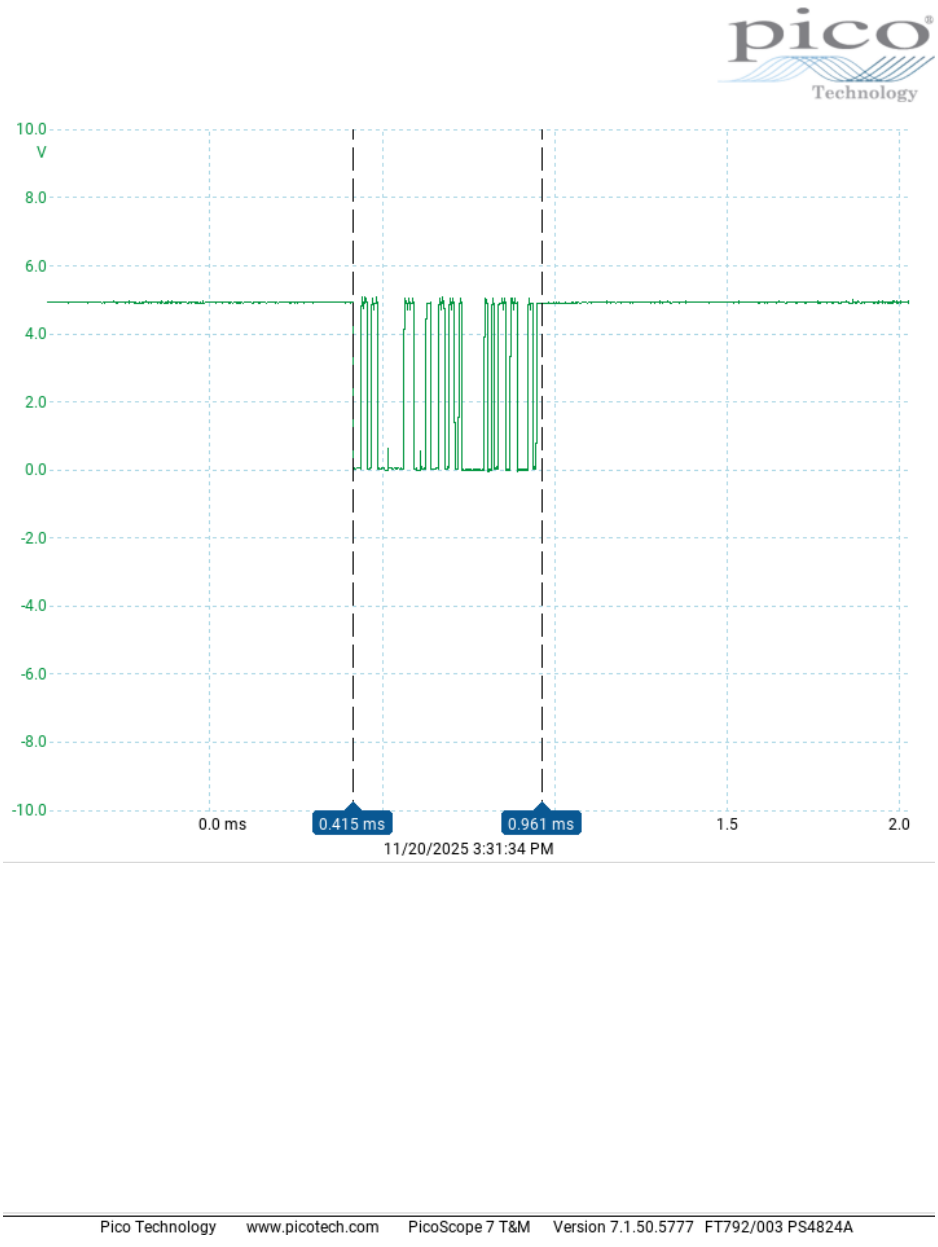

Figure 4 Timing measurement of the angular position reading from the AS5600 encoder.

The green colored signal represents the SDA - which carries the angular position data.

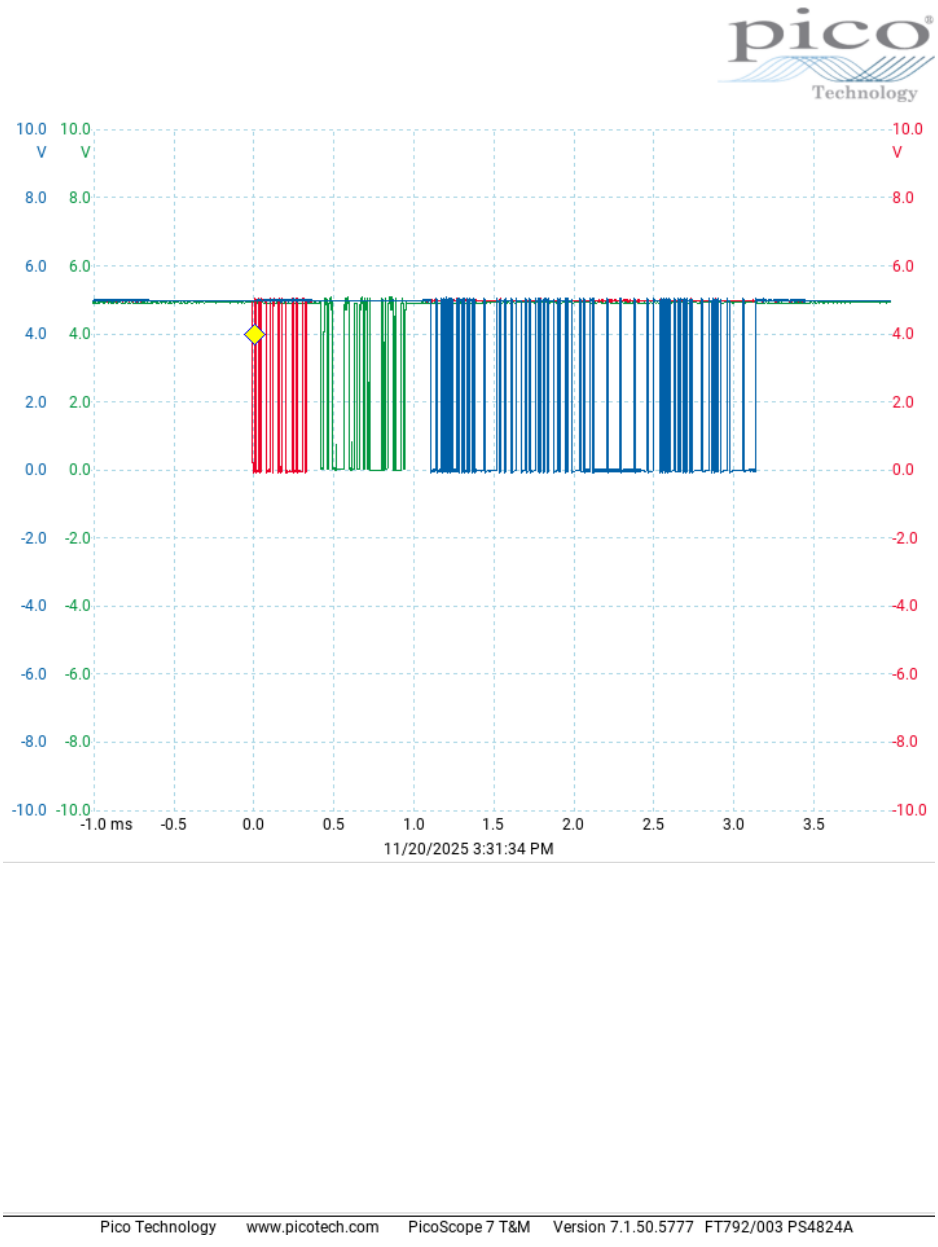

Figure 5 One full communication cycle timing measurement including the encoder read timing, to verify the order of operations.

In the fig. 5 we see the real-time measured order of operations, the red colored signal represents the control input from MATLAB, the green colored signal represents SDA and the blue colored signal is for the data sent to MATLAB respectively. The figure clearly shows the order of operations defined earlier is maintained in real-time.

1.5 Disturbances

The system is subject to various disturbances that can affect its performance and behavior. Categorizing the disturbances into internal and external disturbances, we can identify the following:

- Internal Disturbances:
  - Frictional forces within the motor and pendulum mechanism, which introduce non-linearities and affect the system's response.
  - Electrical noise in the motor driver circuitry, which can lead to fluctuations in the motor's performance.
  - Electrical noise in the sensor readings from the rotary encoder, which introduce inaccuracies in the measured angular position.
- External Disturbances:
  - Variations in supply voltage to the motor, which can affect the motor's torque output.
  - Environmental factors such as temperature changes, which can influence the motor's characteristics and frictional forces.
  - External forces acting on the pendulum, such as air currents or vibrations from the surrounding environment.

The difference between internal and external disturbances lies in their origin. *Internal disturbances* originate from within the system itself, such as friction. *External disturbances*, on the other hand, originate outside of the system, such as variations in supply voltage or manual interference. When designing controllers for the system, it is crucial to consider these disturbances to ensure robust performance and stability under time varying conditions. We can model, predict and compensate for internal disturbances more effectively, because they are inherent to the system's dynamics - allowing for more accurate control strategies. That is if we can separate them from the external disturbances.

While external disturbances are often unpredictable and harder to model, requiring more robust control techniques to mitigate their effects. We can attempt to rid the system of the external disturbances, by conducting the experiments in a controlled environment, where we can minimize the impact of external factors. Another approach is to design the controller to be robust against a certain range of external disturbances, e.g. let us assume the maximum external force acting on the pendulum is known, we

can design the controller to handle this worst-case scenario. The implementation of such a robust control method, can be based on feedback control, where the controller monitors the system's input and output signals, and only enables the robust control mode when it detects significant deviations from the desired behavior - based on the experimental data, indicating the presence of external disturbances.

We will primarily focus on addressing internal disturbances in our controller design, while also considering an added weight to the pendulum as an external disturbance, to evaluate the controller's robustness against such perturbations. Next, we will analyze the correctness of proposed statistical assumptions about the disturbances affecting the system, whether we can detect the deviations of the real-system from the modeled system, caused by this specific external disturbance.

When employing an observer-based controller, we attemp to provide the observer with a model that accounts for the internal disturbances, allowing it to estimate the system's state more accurately. This improved state estimation can help the controller to compensate for the effects of internal disturbances, leading to better overall performance. In our case, we will implement a Kalman filter (KF) and Extented Kalman filter (EKF) as the observers - more about the observers in sec. 4, which are well-suited for handling systems with internal disturbances and measurement noise. They optimally estimate the system's state based on the available measurements and a provided model of the system, taking into account the statistical properties of the measurement and process noise. We will discuss the differences between the two observers in more detail in the sec. 5. At the end of the analysis, we will evaluate the performance of the designed controllers in the presence of both internal and external disturbances, assessing their robustness and effectivness in maintaining the desired system behavior. Additionally, we will analyze the accuracy of the state estimation provided by the observers, comparing the estimated states to the actual system states obtained from experimental data.

1.6 Static Characteristic

Otherwise known as the steady-state characteristic, represents the relationship between the control input and the measurable output when the system is in a steady state. We will abbreviate all the parameters related to steady state with the subscript $(\cdot)_{\text{SS}}$, not to be confused with the SS representation.

1.6.1 Experiment Design

To measure the static characteristic of the system, we require knowledge about the system's inputs and outputs, which we have already defined in the previous sec. 1.1, sec. 1.3 and sec. 1.4. In order to gather the necessary data for constructing the static

characteristic, we will conduct an open-loop experiment where we systematically increase the control input from the minimum value of 0% to the maximum value of 100% in fixed increments. At each increment, we will allow the system to reach a steady state before moving to the next control input value. This ensures that the measured output accuratly reflects the system's response to the provided control input.

It is crucial to determine the appropriate control input increment size and the duration to wait for the system to reach steady state at each step (in case the measurement is automated). A smaller increment in the control input will provide a more detailed static characteristic, but it will also increase the total experiment duration. Conversely, a large increment will reduce the experiment time but is very likely to miss important details about the system's behavior, especially in non-linear regions, e.g. Coloumb friction effects at low control inputs. We will choose an increment size of 1%, which provides a good balance between detail and experiment duration. Because we chose a smaller increment size, we can wait for a shorter duration at each step, as the system will not be subjected to large changes in control input, throughout the mostly linear regions of the static characteristic. We will wait for 10 seconds at each step, which is sufficient for the system to reach steady state. Basing this duration on preliminary experiments conducted prior to the actual measurement. The experiment design is summarized in tab. 5.

Table 5 Static characteristic experiment design

Parameter	Value/Range	Unit
Control input range	0% to 100%	[%]
Control input increment	1%	[%]
Steady state wait time	10	[s]
Total number of steps	101	[steps]

1.6.2 Data Acquisition

To acquire the data for the static characteristic, we have implemented a MATLAB script that automates the entire process. The script is designed with ease-of-use in mind, allowing for quick setup and execution of the experiment. As all the parameters for the experiment are predefined in the script, but can be easily modified to suit different requirements. All the scripts contain specific parts dedicated to defining the experiment parameters, creating default repository folders for storing the logs, error handling in case the program unexpectedly terminates, initializing the communication with the MCU, executing the control input sequence, and logging the measured output data. The script starts by clearing any previous data, figures, and console outputs to ensure a clean

environment for the new experiment. It then creates the necessary folders for storing the logs - if not already present, and initializes all the parameters required for the experiment. After setting up the environment, the script steps into the error handling section, which ensures correct termination of the whole process, thus disconnecting the MCU and saving any acquired data. Once within the error handling section, the script proceeds to initialize the USART communication with the MCU, while setting the correct baud rate and port. After successfully establishing the communication, the script enters the main experiment loop, where all the timing is handled within this loop in MATLAB. The script literally loops infinitely, until it is time to send the next control input value. The correctness of the timing is ensured by using MATLAB's `datetime("now")` function to get the current time, and comparing it to the time when the last control input was sent. When it is time to send the next control input, the script sends the value to the MCU, waits for the response containing the measured output data, and logs the data into a preallocated array for further processing, and in case the experiment is interrupted, not all the data is lost. This process continues until either the stop condition is met (e.g., reaching the allocated time for the experiment), or the experiment is complete (all control input values have been sent and corresponding output data acquired). Afterwards, the script exists the error handling section, ensuring the proper disconnection from the MCU and saving of the acquired data for further analysis. Finally, the script generates plots of the static characteristic, providing the visual confirmation of the measured data. A complete measurement can be seen on fig. 6.

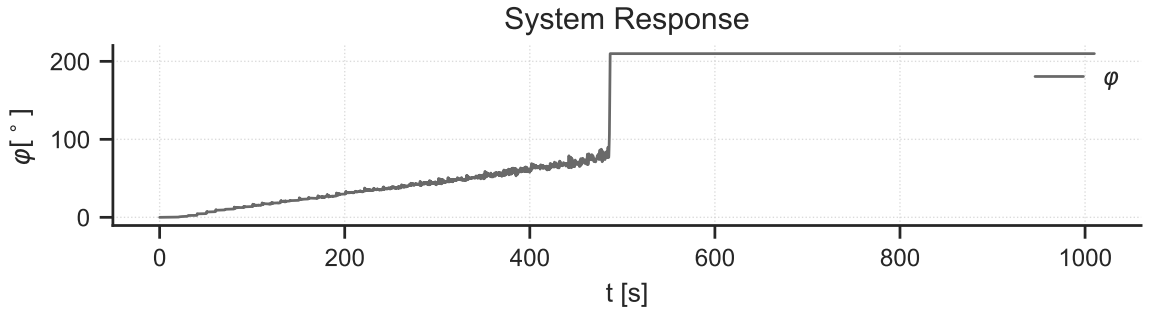

Figure 6 Measured output response in time, for the duration of the static characteristic measurement.

We can see in fig. 6 the system exhibiting nonlinear characteristics, with a deadzone at time near 0s (seconds) and saturation near 500s. We can observe the nonlinearities more clearly in fig. 8, where we zoom into the dead zone and saturation regions of the static characteristic. From the measured data, we can extract the steady-state output

values corresponding to each control input value, which will further be used to construct the static characteristic of the system.

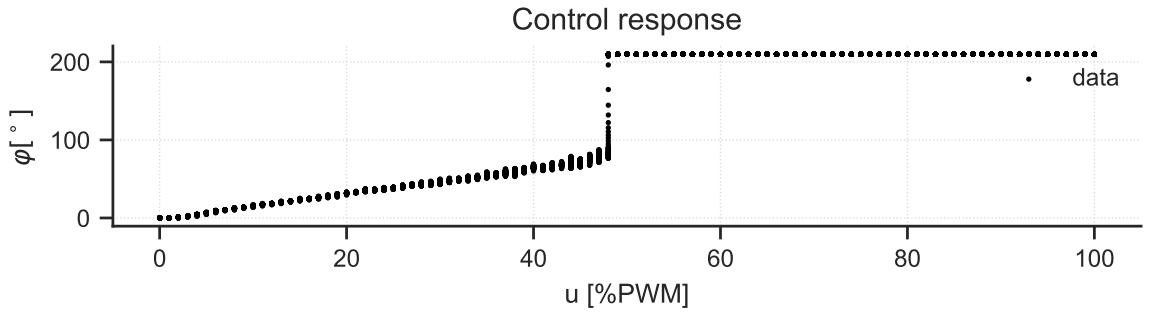

Figure 7 Input-output characteristic of the system.

In the fig. 7 we can see the measured interval gain of the system, because it is before processing the data and extracting the steady-state values. Yet, it outlines the overall shape of the static characteristic, which we will further analyze in the next section. From this plot alone, we can already tell what control input values need to be, to reach certain output angles - useful for feedforward control strategies. Simply said, if we want to hold the pendulum at a certain angle, we can look up the required control input from this plot. If we were to design a feedforward branch for our controller, we could use the static characteristic (see fig. 9) to determine the necessary control input for a desired output angle, e.g. 1-D lookup table with interpolation between the measured points. Allowing the controller to provide the necessary control input directly, without relying solely on feedback from the measured output. This may lead to unstable behavior, if the system is quick to react to changes in positive control input, but slow to react to negative control input requests from the controller. Due to the system's inherent asymmetry, where it can only actively accelerate in one direction (positive), while relying on gravity and friction to decelerate or move in the opposite direction (negative). If we design a very slow feedforward branch, it could still be beneficial for reducing steady-state errors and improving overall system performance, if we do not attempt to use it for aggressive maneuvers, where the system's dynamic characteristic plays a more significant role.

1.6.3 Data Processing

After successfully acquiring the data for the static characteristic, we proceed to process the data to extract the steady-state input/output values. This involves analyzing the logged data to identify the times when the system has reached steady state for each control input value. We can either manually inspect the data to determine the

steady-state values or implement an automated algorithm to detect when the system has stabilized. In our case, we will implement a simple algorithm that calculates the average output value over a predefined time window at the end of each control input step. This approach is effective in filtering out any transient effects and noise, providing a reliable estimate of the steady-state output, because each steady-state output represents statistical average of the output over a certain time window at steady state. Once we have extracted the steady-state input/output pairs, we can plot the static characteristic of the system, which illustrates the relationship between the control input and the measurable output. This plot provides us with valuable insights into the finding linear regions and gains of the system, which can further be used to choose an OP when linearizing the system for controller design. Additionally, we can use the found OP to design an experiment for measuring the dynamic characteristic of the system, which will be discussed in section about Dynamic characteristic.

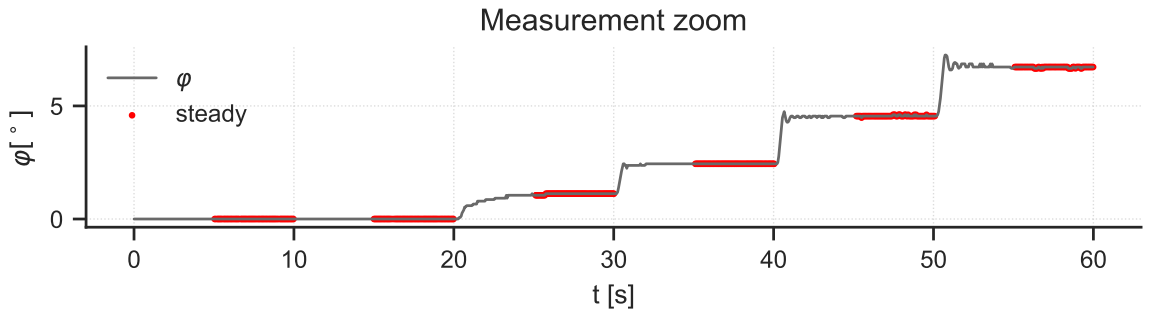

Figure 8 Measured system response of the system, zoomed into the dead zone.

In the zoomed plot of fig. 8, we can clearly see the process of selecting the steady-state output values for each control input step. All the markers in red represent the set of points used to construct the static characteristic of the system.

Finally, we plot the static characteristic of the system in fig. 9, which shows the relationship between the control input and the steady-state output.

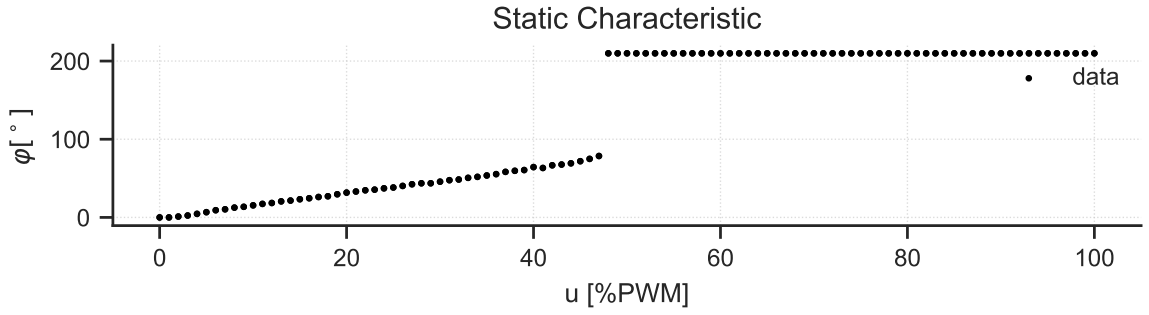

Figure 9 Static characteristic of the system, extracted from the logged data.

1.6.4 Polynomial Model

To gain a better approximation of the static characteristic and a smooth continues function for further analysis, or to replace a lookup table implementation, we can fit a polynomial model to the measured data. We will use MATLAB's built-in polynomial fitting functions to find the best-fitting polynomial of a chosen degree. Simply said, we will use the `polyfit` function to fit a polynomial to the linear part of the measured steady-state input/output pairs. Ranging from 2% to 48% of control input, where the system exhibits a mostly linear behavior. The choice of polynomial degree is crucial, as a higher degree polynomial may fit the data more closely but can also lead to overfitting, while a lower degree polynomial may not capture the characteristics of the system accurately enough. In this case we will choose a 3rd order polynomial (4); which is an overkill for the mostly linear region. We want to show the reader how an overfitted polynomial looks like, thus we choose a higher degree polynomial to illustrate this point. The fitted polynomial model can be expressed mathematically in the following form

$$y(u) = k_3u^3 + k_2u^2 + k_1u + k_0. \tag{4}$$

After fitting the polynomial to the measured data, we can plot the fitted polynomial alongside the measured static characteristic to visually assess the quality of the fit. This is illustrated in fig. 10.

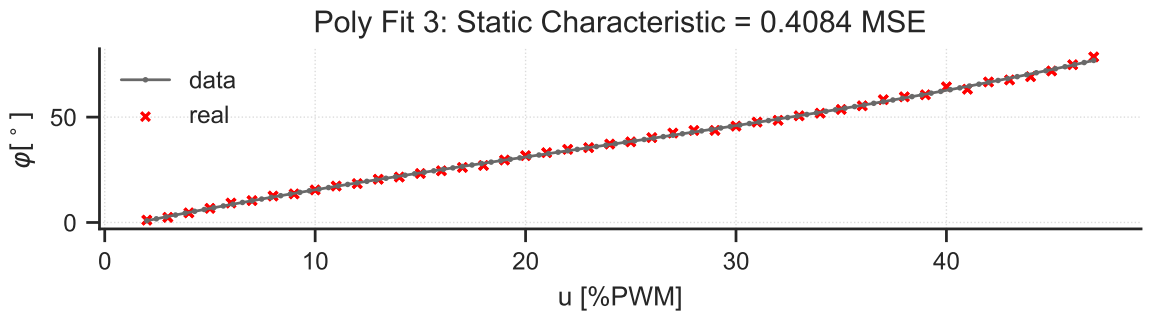

Figure 10 Static characteristic of the system with fitted polynomial model.

The fitted polynomial model provides a smooth approximation of the static characteristic, which can be useful for further analysis and controller design, but is of higher degree than necessary for the mostly linear region of interest. Even though the polynomial fits the data well, with a small mean squared error (MSE), it may not be the most efficient representation for controller design purposes. Because of its higher degree, it may introduce unnecessary complexity into the controller design process. In such cases, a simpler linear model or a lower degree polynomial may be more appropriate, as they can capture the essential nuances of the system's behavior without adding unwanted

complexity. The 3rd order polynomial with substituted coefficients for our system is as follows

$$y(u) = 0.0004u^3 + (-0.0289)u^2 + 2.1281u - 3.3064. \tag{5}$$

Where $y(u)$ is the steady-state output angle in degrees, and u is the control input percentage value. If look at the coefficients of the polynomial, we can see that the cubic and quadratic terms have relatively small coefficients compared to the other terms, indicating that their contribution to the overall output is minimal in the linear region of interest. Thus, we can fit a new polynomial of lower degree, e.g. 1st or 2nd order polynomial, to the same data and compare the results to see if they provide a sufficiently accurate representation of the static characteristic for our controller design purposes. Compare the terms of the newly fitted polynomial and see if the higher degree terms have significantly smaller coefficients, we may attempt to discard them for a simpler model and repeat the process until we reach a satisfactory balance between model accuracy and complexity.

2 System Modeling

Within this section, we will analyze the system's physical modeling. We will derive a physical model of the system using Lagrangian mechanics, which involves defining the kinetic and potential energies of the system, and applying the Euler-Lagrange equation to obtain the equations of motion. The Lagrangian L is defined as the difference between the kinetic energy T and potential energy V of the system:

$$L = T - V \tag{6}$$

The Lagrange's equation in generalized coordinates is given by

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0 \tag{7}$$

where q_i are the generalized coordinates, $\dot{q}_i$ are the generalized velocities.[1] We will use this framework to derive the equations of motion for the pendulum system, taking into account the forces acting on it, including gravitational forces and the torque generated by the motor. The final form of the Lagrange's equation we will be using to derive the equations of motion for our pendulum system is given by adding the generalized forces Q_i to the right-hand side of (7):

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_i \tag{8}$$

Where Q_i represents the generalized forces acting on the system. Which in our case will include the torque generated by the propeller, frictional forces, and any other external forces acting on the pendulum.

2.1 Physical Modeling

Derive the physical model of the pendulum system using Lagrangian mechanics, starting from defining the kinetic and potential energies of the system, and applying the Euler-Lagrange equation to obtain the equations of motion. For now we will present the final derived equation in two forms - a clean form with coefficients, and a expanded form with all the terms shown explicitly. In the second thesis part we will show the whole derivation process. The final clean differential equation of motion for the pendulum system is given by:

$$\ddot{\varphi}(t) = k_1\tau_m(t) - k_2\tau_s(t) - k_3\sin(\varphi(t)) \tag{9}$$

Where the coefficients are defined as:

$$k_1 = \frac{k_{p2m}}{I_T} \tag{10}$$

$$k_{p2m} = \frac{1}{4163} \tag{11}$$

$$k_2 = \frac{1}{I_T} \tag{12}$$

$$k_3 = \frac{G_1}{I_T} \tag{13}$$

$$G_1 = g\left(m_1\frac{L}{2} + m_2(L+R)\right) \tag{14}$$

$$I_T = \frac{1}{3}m_1L^2 + \frac{2}{5}m_2R^2 + m_2(L+R)^2 \tag{15}$$

$$\tag{16}$$

Where:

- φ is the pendulum angle (rad)
- $\dot{\varphi}$ is the angular velocity of the pendulum (rad/s)
- τ_m is the motor torque (Nm)
- τ_s is the total frictional torque (Nm)
- m_1 is the mass of the pendulum rod (kg)
- m_2 is the mass of the pendulum motor (kg)
- L is the length of the pendulum rod (m)
- R is the radius of the pendulum propeller cover (m)
- g is the acceleration due to gravity (m/s²)
- I_T is the total moment of inertia of the pendulum system (kg · m²)
- k_{p2m} is the motor torque constant (Nm per PWM%)

The expanded form of the final differential equation of motion for the pendulum system is given by:

$$\ddot{\varphi}(t) = \frac{1}{I_T}\left(k_{p2m}\tau_m(t) - \beta\dot{\varphi}(t) - \tau_c\tanh(k_c\dot{\varphi}(t)) - k_s\exp\left(-\left[\frac{\dot{\varphi}(t)}{\dot{\varphi}_{brk}}\right]^2\right)\frac{\dot{\varphi}(t)}{\dot{\varphi}_{brk}}\right) - \frac{1}{I_T}\left(k_a\dot{\varphi}^2(t)\tanh(k_r\dot{\varphi}(t)) + G_1\sin(\varphi(t))\right) \tag{17}$$

Where all the variables are as previously defined, and the frictional forces have been expanded to show their individual contributions to the total frictional torque $\tau_s(t)$.

2.2 Frictions

All the frictional forces acting on the pendulum can be modeled as torques opposing the motion of the pendulum, thus we will be adding them along side the other generalized forces Q_i in (8). Within the friction models, we will always consider the direction of the angular velocity $\dot{\varphi}$ to determine the direction of the frictional torque using the smooth continues function hyperbolic tangent (tanh), because of the near zero continuity it provides. The models will always negate the frictional torque to oppose the motion of the pendulum, thus meaning we will always have a negative sign as the first operator in the modeled friction.

2.2.1 Viscous Friction

Viscous friction is a force that opposes the motion of the pendulum and is proportional to its angular velocity and is linear. Allowing for the use of this friction when a linearized model of the system is considered. It can be modeled as:

$$\tau_{viscous} = -\beta\dot{\varphi} \tag{18}$$

where $\tau_{viscous}$ is the viscous friction torque, β is the viscous friction coefficient, and $\dot{\varphi}$ is the angular velocity of the pendulum.

2.2.2 Coulomb Friction

Otherwise known as dry friction. Coulomb friction is a constant force that opposes the motion of the pendulum, regardless of its velocity. It can be modeled as:

$$\tau_{coulomb} = -\tau_c \tanh(k_c\dot{\varphi}) \tag{19}$$

where $\tau_{coulomb}$ is the Coulomb friction torque, τ_c is the Coulomb friction coefficient, and $\tanh(k_c\dot{\varphi})$ is the hyperbolic tangent function of the angular velocity to smooth the transition near zero velocity, k_c is the smoothing coefficient. We adopt an approach with the hyperbolic tangent function to avoid discontinuities in the model at zero velocity and possible noise induced chattering effects when observing the velocity state of the system. Equation (19) can be rewritten into the following form, where we force an interaction between the Coloumb friction and Sticky friction models

$$\tau_{coulomb} = -\tau_c \tanh(h_c \frac{\dot{\varphi}}{\dot{\varphi}_{dry}}) \tag{20}$$

where h_c is the modified smoothing coefficient for Coulomb friction, and $\dot{\varphi}_{dry}$ is the dry angular velocity threshold, which can be further defined as

$$\dot{\varphi}_{dry} = \frac{\dot{\varphi}_{brk}}{10} \tag{21}$$

where $\dot{\varphi}_{brk}$ is the breakaway angular velocity threshold used to define the Stribeck angular velocity in (24), thus creating a relation between the two friction models.

2.2.3 Sticky Friction

Otherwise known as adhesion friction. Sticky friction is a force that opposes the initiation of motion when the pendulum is at rest. It can be modeled as:

$$\tau_{sticky} = -\sqrt{2e}(\tau_{brk} - \tau_{coulomb}) \exp\left(-\left[\frac{\dot{\varphi}}{\dot{\varphi}_S}\right]^2\right) \frac{\dot{\varphi}}{\dot{\varphi}_S} \tag{22}$$

$$\tau_{sticky} = -k_s \exp\left(-\left[\frac{\dot{\varphi}}{\dot{\varphi}_S}\right]^2\right) \frac{\dot{\varphi}}{\dot{\varphi}_S} \tag{23}$$

where τ_{sticky} is the sticky friction torque, e is the Euler number approximation, τ_{brk} is the breakaway friction torque, $\tau_{coulomb}$ is the Coulomb friction torque, $\dot{\varphi}_S$ is the Stribeck angular velocity threshold, and $k_s = \sqrt{2e}(\tau_{brk} - \tau_{coulomb})$ is the sticky friction coefficient.[2, 3] The Stribeck effect describes the phenomenon where the frictional force decreases as the velocity of the pendulum increases from rest, reaching a minimum at a certain velocity before transitioning to Coulomb friction at higher velocities. Stribeck angular velocity threshold $\dot{\varphi}_S$ defines the velocity at which this transition occurs and can be defined as follows

$$\dot{\varphi}_S = \dot{\varphi}_{brk}\sqrt{2} \tag{24}$$

where $\dot{\varphi}_{brk}$ is the breakaway angular velocity threshold, which defines the velocity at which the breakaway friction transitions to Coulomb friction and can be determined experimentally, or estimated based on the system's characteristics.

2.2.4 Air Resistance

Air resistance is a force that opposes the motion of the pendulum due to the drag caused by air, also called the drag force. It can be modeled as:

$$\tau_{air} = -\frac{1}{2} C_d \rho A r^2 |\dot{\varphi}| \dot{\varphi} \tag{25}$$

where τ_{air} is the air resistance torque, C_d is the drag coefficient, ρ is the air density, A is the cross-sectional area of the pendulum, r is the distance from the pivot point to the center of mass, and $\dot{\varphi}$ is the angular velocity of the pendulum.[4] In this model,

the air resistance torque is proportional to the square of the angular velocity, thus we include the absolute value of the angular velocity $|\dot{\varphi}|$ to ensure that the torque always opposes the motion of the pendulum, without needing to rely on the hyperbolic tangent function. We will further simplify (25) to a form proportional to the square of the angular velocity for easier integration into our model. Thus, we can rewrite it as:

$$\tau_{air} = -k_a \dot{\varphi}^2 \tanh(k_r \dot{\varphi}) \tag{26}$$

where k_a is the air resistance coefficient, k_r is the smoothing coefficient and $\dot{\varphi}$ is the angular velocity of the pendulum.

Finally, the total friction torque $\tau_{friction}$ acting on the pendulum can be expressed as the sum of all the individual friction torques:

$$\tau_{friction} = \tau_{viscous} + \tau_{coulomb} + \tau_{sticky} + \tau_{air} \tag{27}$$

$$\tau_{friction} = -\beta\dot{\varphi} - \tau_c \tanh(k_c \dot{\varphi}) - k_s \exp\left(-\left[\frac{\dot{\varphi}}{\dot{\varphi}_{brk}}\right]^2\right) \frac{\dot{\varphi}}{\dot{\varphi}_{brk}} - k_a \dot{\varphi}^2 \tanh(k_r \dot{\varphi}) \tag{28}$$

equation (28) will be used in our modeling of the pendulum system by including it in the generalized forces Q_i in the Lagrange equation (8). The modeled friction will help us better represent the real-world behavior of the pendulum system, accounting for energy losses due to frictional forces acting on the system. A simple preview of the friction model behavior can be seen in fig. 11, where we plot the total friction torque $\tau_{friction}$ as a function of angular velocity $\dot{\varphi}$ using example coefficients for each friction type.

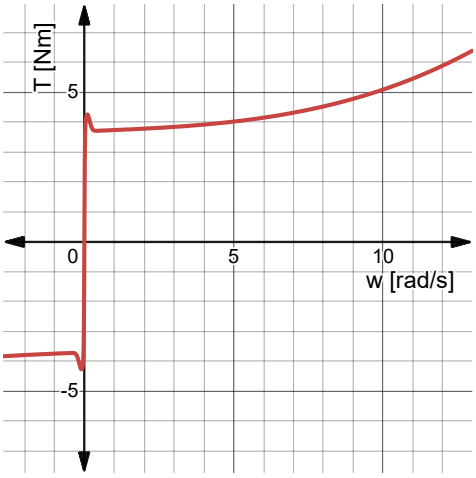

Figure 11 Plotted preview of the total friction torque $\tau_{friction}$ as a function of angular velocity $\dot{\varphi}$ using example coefficients for each friction type.

The coefficients used in the preview plot are as follows:

Table 6 Example coefficient values used in friction model preview

Coefficient	Value	Unit
β	0.04	$\left[\frac{Nm}{rad/s}\right]$
k_a	0.1	$\left[\frac{Nm}{rad^2/s^2}\right]$
k_r	0.01	$\left[\frac{1}{rad/s}\right]$
k_c	1000	$\left[\frac{1}{rad/s}\right]$
τ_c	3.7	$[Nm]$
k_s	3.14	$[Nm]$
$\dot{\varphi}_{brk}$	0.1	$[rad/s]$
τ_{brk}	5.05	$[Nm]$

3 System Representations

Throughout the thesis we will use the following formulations of the system models in state-space representation. All the linearized models will be represented as linear time invariant (LTI) systems, while the original system will be represented using nonlinear state-space models.

3.1 Continues Space-State Model

The generalized continues-time SS representation of an LTI system can be formulated as follows

$$\dot{\mathbf{x}}(t) = A_c\mathbf{x}(t) + B_c\mathbf{u}(t) \tag{29}$$

$$\mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{u}(t) \tag{30}$$

where $\mathbf{x}(t)$ is the state vector at time t, $\mathbf{u}(t)$ is the control input, $\mathbf{y}(t)$ is the measured output, and A_c, B_c, C^{T}, and D are the system matrices derived from the linearized model. For consistency we will use C defined as a row matrix.

3.1.1 Continues Space-State Stochastic Model

The generalized continues-time SS stochastic model of a LTI system can be represented in the following form

$$\dot{\mathbf{x}}(t) = A_c\mathbf{x}(t) + B_c\mathbf{u}(t) + \mathbf{w}(t) \tag{31}$$

$$\mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{u}(t) + \mathbf{v}(t) \tag{32}$$

where $\mathbf{w}(t)$ is the process noise and $\mathbf{v}(t)$ is the measurement noise, both assumed to be zero-mean Gaussian noise with known covariance matrices. The stochastic model accounts for uncertainties in the system dynamics and measurements, making the model more realistic when simulating real-world scenarios. The implementation process is straight forward, as it only requires adding noise terms to the state transition and measurement equations. Respectively, the process noise $\mathbf{w}(t)$ and measurement noise $\mathbf{v}(t)$ definition used within the thesis can be formulated using the Gaussian distribution as follows:

$$\mathbf{w}(t) \sim \mathcal{N}(0, Q) \tag{33}$$

$$\mathbf{v}(t) \sim \mathcal{N}(0, R) \tag{34}$$

where $Q \in \mathbb{R}^{n\times n}$ is the process noise covariance matrix, $R \in \mathbb{R}^{m\times m}$ is the measurement noise covariance matrix, n is the number of states of the system and m is the number of

measurements. These matrices define the statistical properties of the noise affecting the system and measurements, respectively, and are crucial for the design and performance of state estimators such as the KF and EKF. We can easily model Normal (Gaussian) noise using the following function:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{35}$$

where μ is the mean (expected value) and σ^2 is the variance of the distribution. In our case, both μ values are set to zero, representing zero-mean noise.

$$\mu = E[X] = \frac{1}{N} \sum_{i=1}^{N} X_i \tag{36}$$

where $E[X]$ is the expected value of the random variable $X \sim \mathcal{N}(\mu, \sigma^2)$, $N \in \mathbb{N}$ is the total number of samples, and $\mu \in \mathbb{R}$ is the mean. The variance σ^2 is calculated as:

$$\sigma^2 = E[(X - \mu)^2] = \frac{1}{N} \sum_{i=1}^{N} (X_i - \mu)^2 \tag{37}$$

$$\sigma = \sqrt{\sigma^2} \tag{38}$$

where $\sigma \in \mathbb{R}^+$ is the standard deviation, which quantifies the amount of variation or dispersion of a set of values around the mean. As an example, if we want to simulate process noise with a standard deviation of 0.3 the variance σ^2 would be 0.09 based on (37). The graphed representation of such noise is shown in Figure fig. 12.

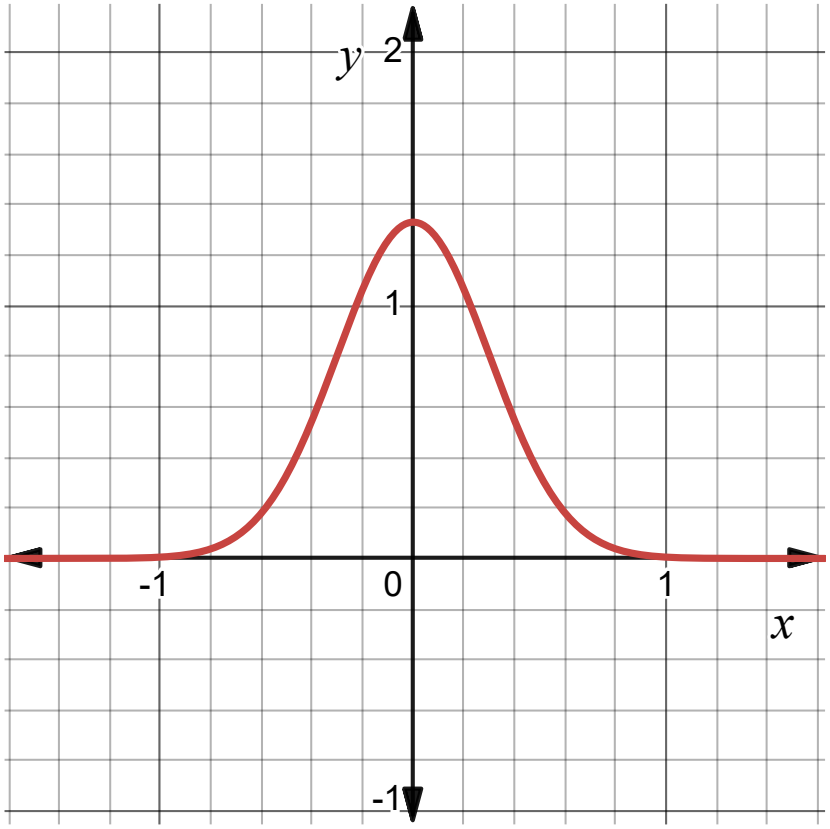

Figure 12 Graphical representation of Gaussian distribution with $\mu = 0$ and $\sigma = 0.3$.

The process of integrating these noise components into the following state-space models is essentially the same, as shown in equations (31) and (32). Except when dealing with discrete-time models, where the noise terms need to be appropriately scaled based on the sampling period to ensure accurate representation of the noise characteristics in the discrete domain. A change of notation is in order when tranforming continuous time to discrete time is required, thus instead of adding $\mathbf{w}(t)$ and $\mathbf{v}(t)$, we add $\mathbf{w}_k$ and $\mathbf{v}_k$ to the respective equations.

3.2 Discrete State-Space Model

The discrete-time SS representation of an LTI system can be formulated as follows

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k \tag{39}$$

$$\mathbf{y}_k = C\mathbf{x}_k + D\mathbf{u}_k \tag{40}$$

where $\mathbf{x}_k$ is the state vector at discrete time step k, $\mathbf{u}_k$ is the control input, $\mathbf{y}_k$ is the measured output, and A, B, C^{T}, and D are the system matrices derived from the linearized model.

3.3 Nonlinear Continues State-Space Model

The nonlinear continues-time SS model of the system can be represented as:

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}(t), \mathbf{u}(t)) \tag{41}$$

$$\mathbf{y}(t) = h(t, \mathbf{x}(t), \mathbf{u}(t)) \tag{42}$$

where f and h are nonlinear functions that describe the system dynamics and output, respectively.

3.4 Nonlinear Discrete State-Space Model

The nonlinear discrete-time SS model of the system can be represented as:

$$\mathbf{x}_{k+1} = f(k, \mathbf{x}_k, \mathbf{u}_k) \tag{43}$$

$$\mathbf{y}_k = h(k, \mathbf{x}_k, \mathbf{u}_k) \tag{44}$$

where f and h are nonlinear functions that describe the system dynamics and output, respectively.

4 Observability and Controllability

To ensure that we can estimate the states of the system accurately, we will perform an observability analysis for the derived model of the pendulum system.

4.1 Observability of linear systems

For linear systems, we can use the observability matrix to mathematically determine if the system is observable. The observability matrix $\mathcal{O}$ is defined as:

$$\mathcal{O} \equiv \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{45}$$

Where A is the state matrix, C is the output matrix and n is the number of states - also called the order of the system, dimensionality.[5–7]

A linear system is considered observable if the observability matrix is of rank equal to the number of states n. A mathematical representation of this condition is given by:

$$\text{rank}(\mathcal{O}) = n \tag{46}$$

Otherwise, if the rank of the observability matrix is less than n, further analysis is required to determine which states are observable. Thesis [7] proposes method called *Kalman Decomposition* to decompose the system into observable and unobservable subspaces. We will further prove the observability of our linearized pendulum system in Chapter sec. 5.1 when we design the state observer, using the equations (45) and (46).

4.2 Observability of nonlinear systems

For nonlinear systems, the observability analysis is more complex. An approach exists to the problem using the generalized ratio condition of circuit theory and the condition of positive definitness; described in the following paper [8], relying on Taylor series expansion applied to the measurement function around the initial state and time. We chose a similar approach of applying Lie derivatives to analyze the observability of nonlinear systems as proposed in articles [6, 9]. Let us consider a nonlinear system - with no input, represented by Nonlinear state-space (NSS) form, state transition given as

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}) \tag{47}$$

and measurement function

$$\mathbf{y} = \mathbf{h}(t, \mathbf{x}) \tag{48}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state vector, $\mathbf{y} \in \mathbb{R}^m$ is the output vector, $\mathbf{f} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ is a smooth vector field representing the system dynamics, and $\mathbf{h} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^m$ is a smooth function representing the output. To analyze the observability of this nonlinear system, we will use Lie derivatives of the measurement function to construct the observability matrix $\mathcal{O}$ as explicitly stated in [6, 9]. The Lie derivative of a function $\mathbf{y}$ along a vector field $\mathbf{f}$ is defined as:

$$\begin{aligned}
\pounds_{\mathbf{f}}^0(\mathbf{y}(\mathbf{x})) &= \mathbf{y}(\mathbf{x}) \\
\pounds_{\mathbf{f}}^1(\mathbf{y}(\mathbf{x})) &= \frac{\partial \mathbf{y}(\mathbf{x})}{\partial \mathbf{x}} \cdot \mathbf{f}(\mathbf{x}) = \nabla_{\mathbf{x}} \pounds_{\mathbf{f}}^0(\mathbf{y}(\mathbf{x})) \cdot \mathbf{f}(\mathbf{x}) \\
\pounds_{\mathbf{f}}^2(\mathbf{y}(\mathbf{x})) &= \frac{\partial}{\partial \mathbf{x}} \left(\pounds_{\mathbf{f}}^1(\mathbf{y}(\mathbf{x})) \right) \cdot \mathbf{f}(\mathbf{x}) = \nabla_{\mathbf{x}} \pounds_{\mathbf{f}}^1(\mathbf{y}(\mathbf{x})) \cdot \mathbf{f}(\mathbf{x}) \\
&\vdots \\
\pounds_{\mathbf{f}}^n(\mathbf{y}(\mathbf{x})) &= \frac{\partial}{\partial \mathbf{x}} \left(\pounds_{\mathbf{f}}^{n-1}(\mathbf{y}(\mathbf{x})) \right) \cdot \mathbf{f}(\mathbf{x}) = \nabla_{\mathbf{x}} \pounds_{\mathbf{f}}^{n-1}(\mathbf{y}(\mathbf{x})) \cdot \mathbf{f}(\mathbf{x})
\end{aligned} \tag{49}$$

Using the Lie derivatives defined in (49), we can construct the observability mapping ϕ for the nonlinear system as follows:

$$\phi = \begin{bmatrix} \pounds_{\mathbf{f}}^0(\mathbf{y}(\mathbf{x})) \\ \pounds_{\mathbf{f}}^1(\mathbf{y}(\mathbf{x})) \\ \pounds_{\mathbf{f}}^2(\mathbf{y}(\mathbf{x})) \\ \vdots \\ \pounds_{\mathbf{f}}^{n-1}(\mathbf{y}(\mathbf{x})) \end{bmatrix} \tag{50}$$

The observability matrix $\mathcal{O}$ for the nonlinear system is then given by the Jacobian of the observability mapping ϕ with respect to the state vector $\mathbf{x}$:

$$\begin{aligned}
\mathcal{O} \equiv \frac{\partial \phi}{\partial \mathbf{x}} = \nabla_{\mathbf{x}} \phi &= \begin{bmatrix} \nabla_{\mathbf{x}} \pounds_{\mathbf{f}}^0(\mathbf{y}(\mathbf{x})) \\ \nabla_{\mathbf{x}} \pounds_{\mathbf{f}}^1(\mathbf{y}(\mathbf{x})) \\ \nabla_{\mathbf{x}} \pounds_{\mathbf{f}}^2(\mathbf{y}(\mathbf{x})) \\ \vdots \\ \nabla_{\mathbf{x}} \pounds_{\mathbf{f}}^{n-1}(\mathbf{y}(\mathbf{x})) \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial}{\partial x_1} \left(\pounds_{\mathbf{f}}^0(\mathbf{y}(\mathbf{x})) \right) & \dots & \frac{\partial}{\partial x_n} \left(\pounds_{\mathbf{f}}^0(\mathbf{y}(\mathbf{x})) \right) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1} \left(\pounds_{\mathbf{f}}^{n-1}(\mathbf{y}(\mathbf{x})) \right) & \dots & \frac{\partial}{\partial x_n} \left(\pounds_{\mathbf{f}}^{n-1}(\mathbf{y}(\mathbf{x})) \right) \end{bmatrix}
\end{aligned} \tag{51}$$

When the observability matrix $\mathcal{O}$ defined in (51) is of rank equal to the number of states n, thus satisfying the condition mentioned in (46), the nonlinear system is considered locally observable around the point of linearization.

4.3 Controllability of linear systems

Controllability is a property of a system that determines whether it is possible to drive the system's state from initial state to any desired final state within system's constraints using appropriate control inputs and in finite time. In our case of the pendulum system a single control input is available acting on the system in a specific direction, thus the system inherently containing uncontrollable states.

Imagine the pendelum being in an inital state of angular position as $\theta = 210°$ and angular velocity $\dot{\theta} = 0$ rad/s, it is physically impossible to provide any control input to drive the pendulum from this initial state - let us call such a state a deadlock state. For linear systems, we can use the controllability matrix to mathematically determine if the system is controllable. The controllability matrix $\mathcal{C}$ is defined as:

$$\mathcal{C} \equiv \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix} \tag{52}$$

Where A is the state matrix, B is the input matrix and n is the number of states.[5–7, 9] A linear system is considered controllable if the controllability matrix is of rank equal to the number of states n. A mathematical representation of this condition is given by:

$$\text{rank}(\mathcal{C}) = n \tag{53}$$

Otherwise, if the rank of the controllability matrix is less than n, system is deemed uncontrollable. We will further prove the controllability of our linearized pendulum system in Chapter sec. 5.1 when we design the state observer, using the equations (52) and (53).

4.4 System's Observability Analysis

Before implementing the state observer, we need to ensure that the system is observable. Observability is a property that determines whether the internal states of a system can be determined from its outputs. We will perform an observability analysis for both the linearized and non-linear models of the system as described in sec. 4.

5 Controller Design

In this chapter we will describe the process of designing a controller for the system using the derived non-linear model. We will discuss the selection of the control strategy, the design of the controller, and the tuning of its parameters.

5.1 State Observers

To estimate the states of the system, we will implement a state observer. Given the non-linear nature of the system, we will consider using an EKF as our state observer. While also evaluating the performance of a standard KF for comparison.

5.1.1 Kalman Filter

The KF is an optimal state estimator for linear systems with noise characterized by Gaussian distribution. It uses a series of measurements observed over time, containing the measurement noise and produces estimates of observable states, while incorporating the process noise into the estimation. KF operates in a two-step process - prediction and update, as described in [7]. The steps of the KF algorithm are as follows:

- **Prediction Step:** In this step, the filter predicts the next state of the system and the state error covariance matrix based on the current state estimate and the identified linearized system model.
    1. State Prediction: The filter uses the system model to predict the next state using the current state estimate and control input, simply by applying the SS transition equations.
    2. Error Covariance Prediction: The filter predicts the error covariance matrix, which represents the uncertainty in the state estimate, the lower this value the more certain the estimate is.
- **Update Step:** In this step, the filter updates the Kalman gain, state estimate, output estimate and error covariance matrix using the new measurement data, taking into account the uncertainty in both the prediction and the measurement.
    1. Kalman Gain Calculation: The filter calculates the Kalman gain, which determines how much weight to give the new measurement in comparison to the prediction.
    2. State Update: The filter updates the state estimate by combining the predicted state and the error between the predicted output and the actual measurement, weighted by the Kalman gain.

3. Output Update: The filter updates the output estimate based on the updated state estimate.
4. Error Covariance Update: The filter updates the error covariance matrix to reflect the reduced uncertainty after incorporating the new measurement.

Let us provide the mathematical formulation of the KF algorithm.
The KF algorithm consists of the following steps:

- **Prediction Step:**

$$\hat{\mathbf{x}}_{k|k-1} = A\hat{\mathbf{x}}_{k-1|k-1} + Bu_{k-1} \tag{54}$$
$$P_{k|k-1} = AP_{k-1|k-1}A^{\mathrm{T}} + Q \tag{55}$$

- **Update Step:**

$$S_k = CP_{k|k-1}C^{\mathrm{T}} + R \tag{56}$$
$$K_k = P_{k|k-1}C^{\mathrm{T}}S_k^{-1} \tag{57}$$
$$e_k = y_k - C^{\mathrm{T}}\hat{\mathbf{x}}_{k|k-1} \tag{58}$$
$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + K_k e_k \tag{59}$$
$$P_{k|k} = (I - K_k C)P_{k|k-1} \tag{60}$$

Where:

- $\hat{\mathbf{x}}_{k|k-1}$ is the predicted state estimate at time step k given observations up to time step $k-1$.
- $P_{k|k-1}$ is the predicted error covariance matrix at time step k.
- Q is the process noise covariance matrix.
- S_k is the innovation covariance matrix.
- K_k is the Kalman gain at time step k.
- e_k is the innovation or measurement residual at time step k.
- R is the measurement noise covariance matrix.
- I is the identity matrix.
- $\hat{\mathbf{x}}_{k|k}$ is the updated state estimate at time step k after incorporating the measurement.

- $P_{k|k}$ is the updated error covariance matrix at time step k.
- y_k is the measurement at time step k.

Furthermore, we can rewrite the equation (57) for Kalman gain K_k to a form that can be more computationally stable, because it avoids the direct inversion of the innovation covariance matrix S_k:

$$K_k = ((S_k^{-1})^{\mathrm{T}} C P_{k|k-1}^{\mathrm{T}})^{\mathrm{T}} \tag{61}$$

This form leverages the properties of matrix transposition and inversion to improve numerical stability during computation, especially when dealing with ill-conditioned matrices. This approach is particularly useful in practical implementations of the Kalman filter where numerical precision is a concern and when working with low valued covariance matrices. Because to provided form in (61) can be solved using linear system solvers that are generally more stable than direct matrix inversion.

Conclusion

We have sucecssfully finished 83.33% from the very first task listed in the tasks. We have completed the detailed system description, analyzed the system's inputs and outputs, proposed and conducted experiments to measure the static characteristic, we successfully extracted the system's static characteristic from the measured data. We touched upon the possibility of practical aspects of implementing the control algorithms on a MCU, however we did not go into detail about the implementation itself, because we have yet to design the control system, but we have laid the groundwork for it by measuring the order of communication delays, between the MCU and MATLAB in sec. 1.2.1. This section showcases the functionality of the implemented communication protocol and the logic behind the order of operations, ensuring that the control inputs are sent and the output data is received in a timely manner.

We prepared the ground for system identification by presenting the model representations to be used in the identification process, including the control design process. Upon completing the physical modeling of the system using the Lagrange equations, we obtained a nonlinear model of the system, which can be linearized around an OP to derive a linearized SS model, thus assessing the possibilities for mathematical modeling of the system as 100% complete. Furthermore, we took a step further and presented different types of friction models that can be used to enhance the accuracy of the derived models.

we have outlined the goals and objectives of this thesis in the introduction, which we are steadily progressing towards. We have mentioned control strategies such as LQR and MPC, both of which we have successfully implemented in MATLAB for out system and tested on the real system.

6 Future Work

We will present the results of the control design and implementation in the upcoming chapters, or unfinished chapters. We have laid some of the groundwork needed for the controller design, by discussing state observers such as the KF and EKF, which will be used to estimate the states of the system for feedback in the control loop. Having reviewed and described the KF algorithm in sec. 5.1, we are now prepared to implement it in the control system. Furthermore, we will also implement the EKF using the identified nonlinear model to compare its performance against the standard KF.

In the next chapters we will add the missing pieces of the puzzle, such as the step response, impulse response, free swing response (nonlinear model parameters identification) to be used in the system identification process, and finally we will design and implement the controllers using the derived models, proposed controller strategies in the introduction and state estimators. Experiments concerning the data required for the dynamic characteristic, impulse characteristic and free swing response have been concluded. We just need to process the data and extract the required information from it.

At the same time, we will attempt to identify the noise characteristics of the system, which will be used to tune the process and measurement noise covariance matrices Q and R of the KF and EKF. Physical model of the system has been derived, while this report only presents the final equations, the complete derivation process is documented in paper form, which can be provided upon request, we will be adding the derivation process into the thesis instead of the final equations only.

An attempt to add reactive physical constraints model to the simulation will be made (without saturating the angular position of the system), such as angular limits of the pendulum, but model it by either a stiff spring-damper system at the limits, or by simply inversing the velocity with a damping factor when the limits are hit. All the necessary MATLAB scripts for all the controllers mentioned in the introduction are ready and tested with an already identified linearized model of the system, will be adding the implementation details and simulation results into the thesis as well. We have already tested the LQR + KF on the real system, and it performed satisfactorily, implemented direcrly on an MCU.

Only one controller design remains to be implemented and tested, the Nonlinear model predictive control (NMPC), which we will be working on in the upcoming weeks, after successfully integrating the newly identified physical model into the MPC + EKF and LQR + EKF frameworks. After successful integration in the mentioned control

framework, we can move onto implementing a new control system, we have no prior experience with NMPC, thus this will be a learning experience for us as well.

Bibliography

1. WIDNALL, S. Lecture L20 - Energy Methods: Lagrange's Equations. In: *Dynamics—MIT Course No. 16.17*. Cambridge MA: Massachusetts Institute of Technology, 2009. Available also from: https://ocw.mit.edu/courses/16-07-dynamics-fall-2009/b39e882f1524a0f6a98553ee33ea6f35_MIT16_07F09_Lec20.pdf. MIT OpenCourseWare.
2. ALTPETER, F. *Friction Modeling, Identification and Compensation*. 1999. PhD thesis. Swiss Federal Institute of Technology in Lausanne.
3. QUEIROZ, M. S. de; DAWSON, D. M.; NAGARKATTI, S. P.; ZHANG, F. Control Techniques for Friction Compensation. In: *Lyapunov-Based Control of Mechanical Systems*. Springer, 2000.
4. BENSON, T. *The drag equation*. NASA, 1999. Available also from: https://www.grc.nasa.gov/www/k-12/VirtualAero/BottleRocket/airplane/drageq.html.
5. KAILATH, T. Linear Systems. In: Prentice-Hall, 1980. Information and System Sciences Series. ISBN 9780135369616. Available also from: https://books.google.sk/books?id=ggYqAQAAMAAJ.
6. WHALEN, A. J.; BRENNAN, S. N.; SAUER, T. D.; SCHIFF, S. J. Observability and controllability of nonlinear networks: The role of symmetry. In: American Physical Society (APS), 2015, vol. 5. No. 1.
7. IQBAL, A. *Applications of an Extended Kalman Filter in nonlinear mechanics*. 2019. MA thesis. University Of Management and Technology.
8. KOU, S. R.; ELLIOTT, D. L.; TARN, T. J. Observability of nonlinear systems. *Information and Control*. 1973, **22**(1), 89–99. ISSN 0019-9958. Available from DOI: https://doi.org/10.1016/S0019-9958(73)90508-1.
9. HERMANN, R.; KRENER, A. Nonlinear controllability and observability. *IEEE Transactions on Automatic Control*. 1977, **22**(5), 728–740. Available from DOI: 10.1109/TAC.1977.1101601.