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

AS	AeroShield
DC	Direct Current
GPIO	General-Purpose Input/Output
I2C	Inter-Integrated Circuit
LQR	Linear Quadratic Regulator
MCU	Microcontroller Unit
MPC	Model Predictive Control
NSS	Non-Linear State Space
OP	Operating Point
PWM	Pulse-Width Modulation
SCL	Serial Clock Line
SDA	Serial Data Line
SS	State Space
UART	Universal Asynchronous 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.

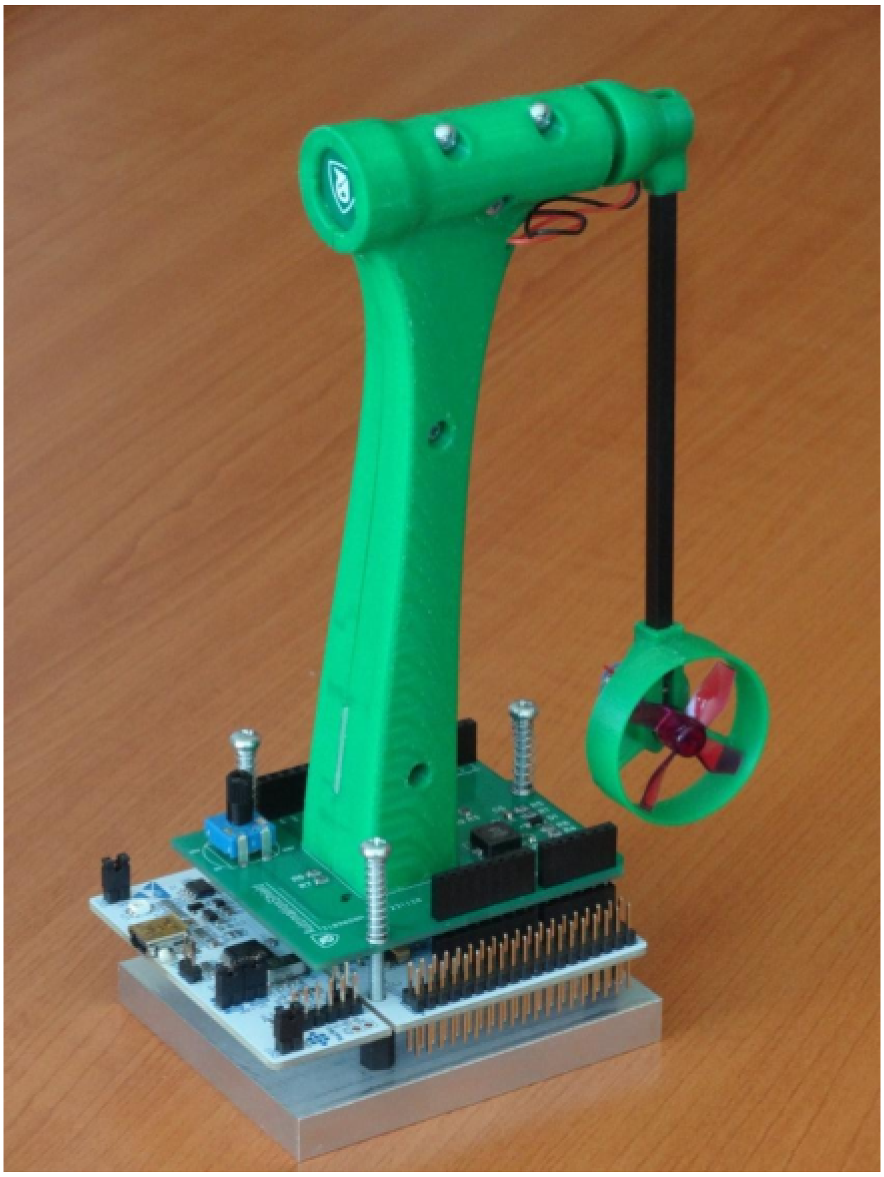

Figure 1 The considered actuated pendulum system named AeroShield (AS) device.

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 UART 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 250000 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.

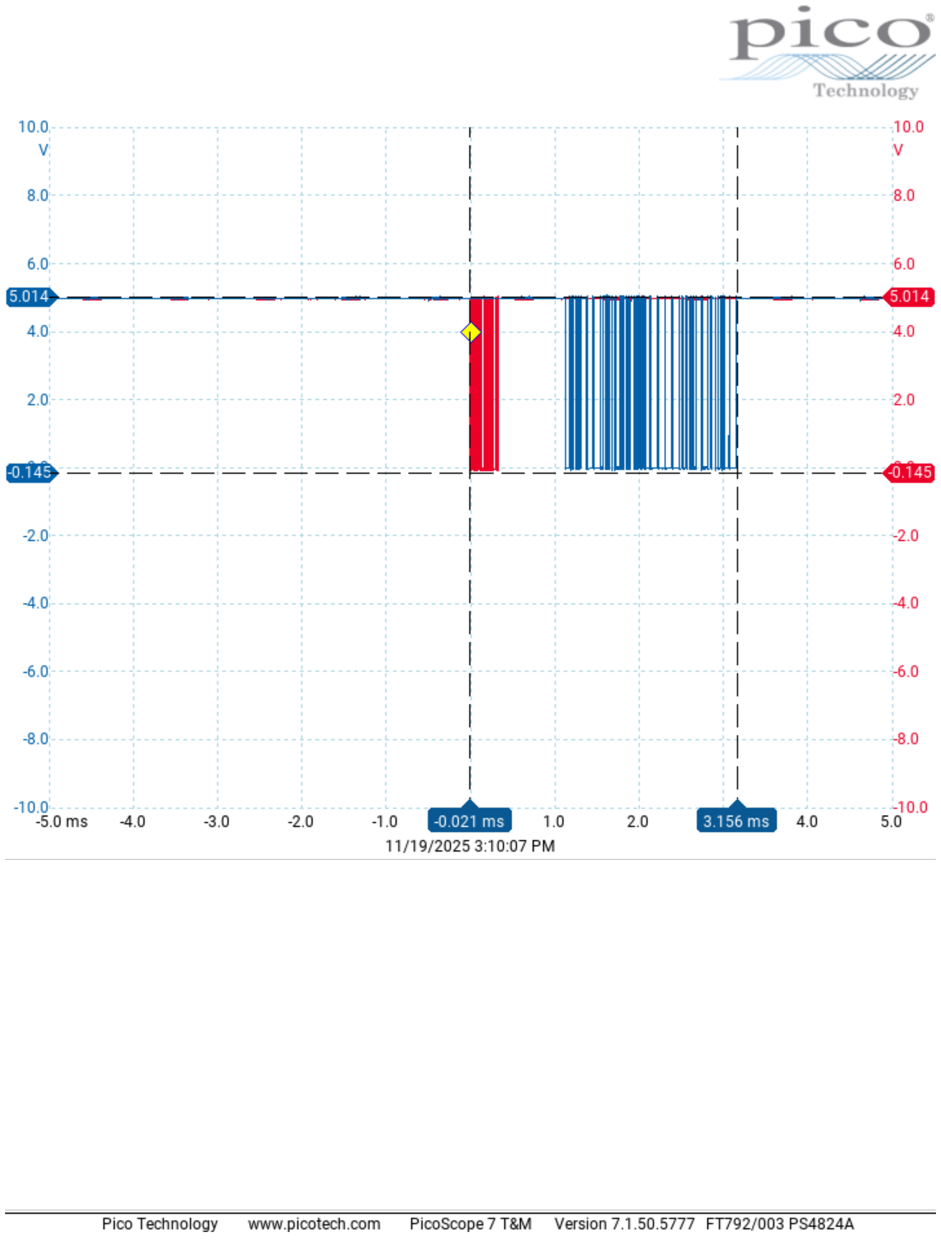

Figure 2 Communication process between the MCU and MATLAB.

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.

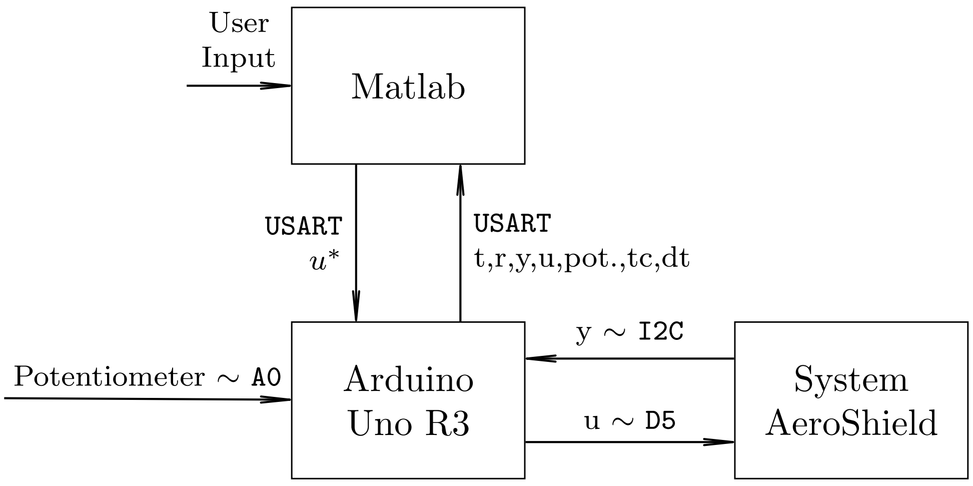

Figure 3 Schematic diagram of the communication protocol implemented for the AS system.

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 precisly 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.

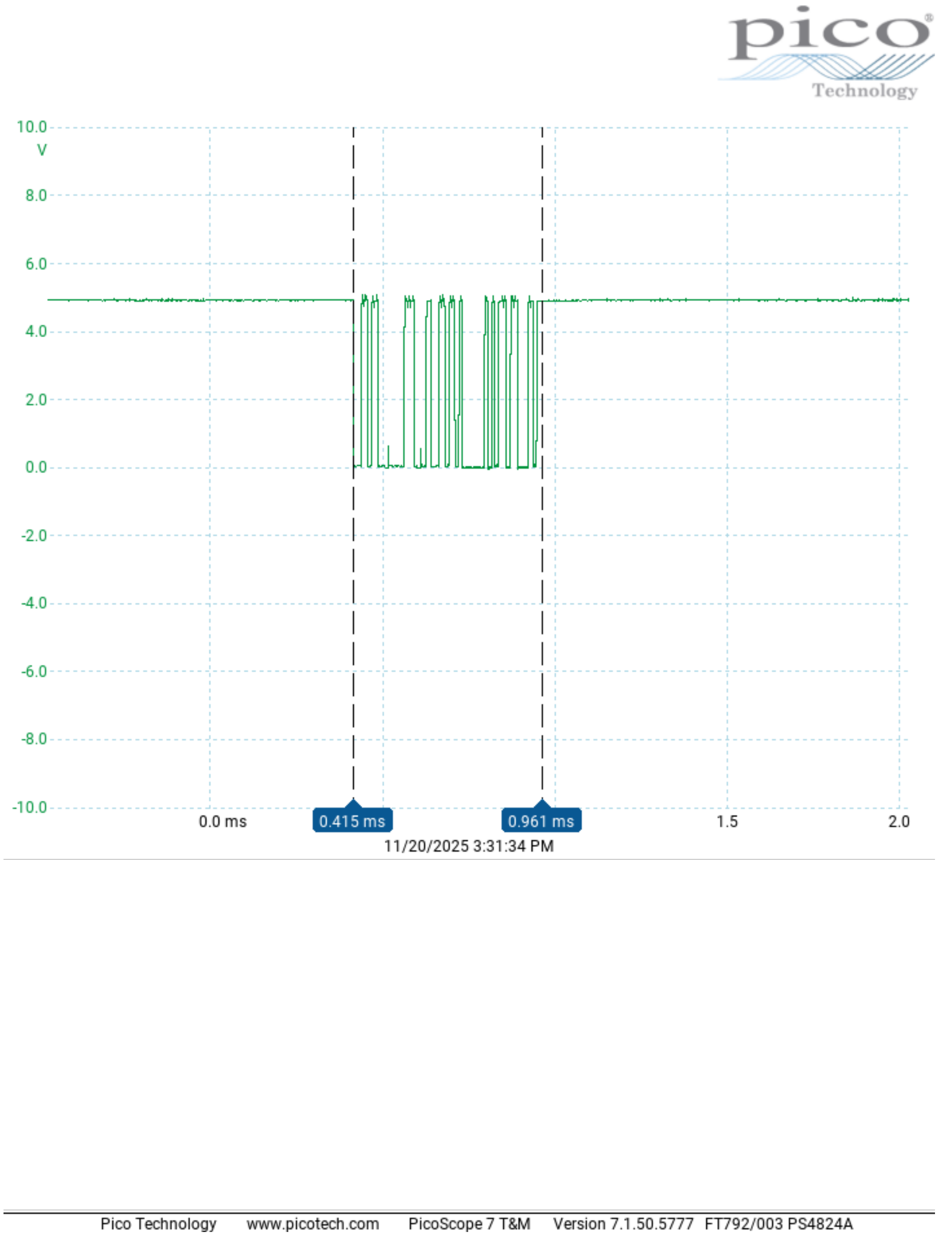

Figure 4 Timing measurement of the angular position reading from the AS5600 encoder.

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

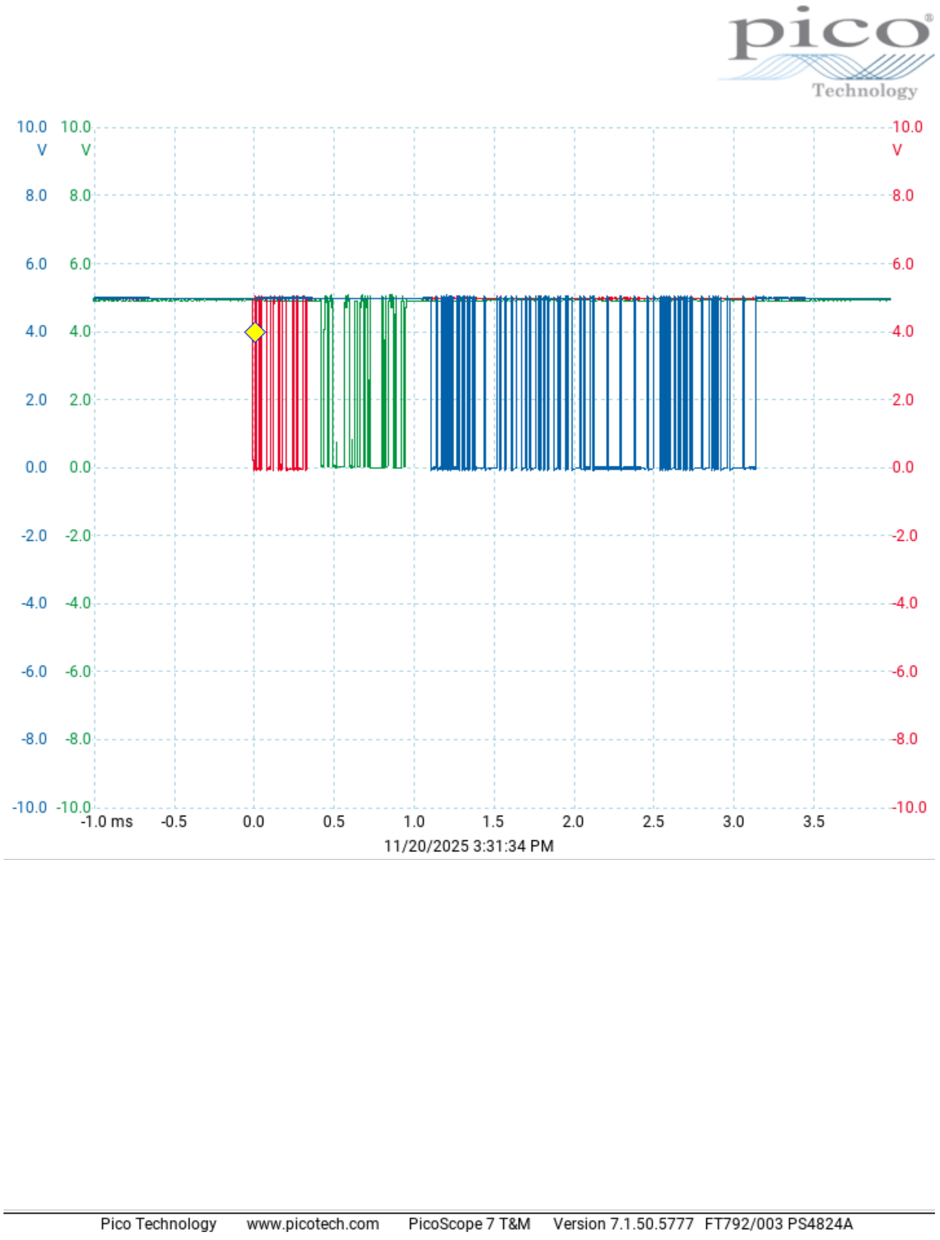

Figure 5 The whole communication cycle timing measurement including the encoder read timing.

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

1.6 Static Characteristic

1.7 Dynamic Characteristic

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:

$$L = T - V \tag{4}$$

The Lagrange's equation in generalized coordinates is given by

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0 \tag{5}$$

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 (5):

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_i \tag{6}$$

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.

TODO: CONTINUE FROM HERE...

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 (6). Within the friction models, we will always consider the direction of the angular velocity $\dot{\theta}$ 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:

$$\tau_{viscous} = -\beta\dot{\theta} \tag{7}$$

where $\tau_{viscous}$ is the viscous friction torque, β is the viscous friction coefficient, and $\dot{\theta}$ 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:

$$\tau_{coulomb} = -\tau_c \tanh(k_c\dot{\theta}) \tag{8}$$

where $\tau_{coulomb}$ is the Coulomb friction torque, τ_c is the Coulomb friction coefficient, and $\tanh(k_c\dot{\theta})$ 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 (8) can be rewritten into the following form, where we force an interaction between the Coloumb friction and Sticky friction models

$$\tau_{coulomb} = -\tau_c \tanh(h_c\frac{\dot{\theta}}{\dot{\theta}_{dry}}) \tag{9}$$

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

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

where $\dot{\theta}_{brk}$ is the breakaway angular velocity threshold used to define the Stribeck angular velocity in (13), 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:

$$\tau_{sticky} = -\sqrt{2e}(\tau_{brk} - \tau_{coulomb}) \exp\left(-\left[\frac{\dot{\theta}}{\dot{\theta}_S}\right]^2\right)\frac{\dot{\theta}}{\dot{\theta}_S} \tag{11}$$

$$\tau_{sticky} = -k_s \exp\left(-\left[\frac{\dot{\theta}}{\dot{\theta}_S}\right]^2\right)\frac{\dot{\theta}}{\dot{\theta}_S} \tag{12}$$

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, $\dot{\theta}_S$ 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 $\dot{\theta}_S$ defines the velocity at which this transition occurs and can be defined as follows

$$\dot{\theta}_S = \dot{\theta}_{brk}\sqrt{2} \tag{13}$$

where $\dot{\theta}_{brk}$ 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:

$$\tau_{air} = -\frac{1}{2}C_d \rho A r^2 |\dot{\theta}|\dot{\theta} \tag{14}$$

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 $\dot{\theta}$ 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 $|\dot{\theta}|$ 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 (14) to a form proportional to the square of the angular velocity for easier integration into our model. Thus, we can rewrite it as:

$$\tau_{air} = -k_a \dot{\theta}^2 \tanh(k_r \dot{\theta}) \tag{15}$$

where k_a is the air resistance coefficient, k_r is the smoothing coefficient and $\dot{\theta}$ 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:

$$\tau_{friction} = \tau_{viscous} + \tau_{coulomb} + \tau_{sticky} + \tau_{air} \tag{16}$$

$$\tau_{friction} = -\beta\dot{\theta} - \tau_c \tanh(k_c\dot{\theta}) - k_s \exp\left(-\left[\frac{\dot{\theta}}{\dot{\theta}_{brk}}\right]^2\right)\frac{\dot{\theta}}{\dot{\theta}_{brk}} - k_a\dot{\theta}^2 \tanh(k_r\dot{\theta}) \tag{17}$$

equation (17) will be used in our modeling of the pendulum system by including it in the generalized forces Q_i in the Lagrange equation (6). 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. 6, where we plot the total friction torque $\tau_{friction}$ as a function of angular velocity $\dot{\theta}$ using example coefficients for each friction type.

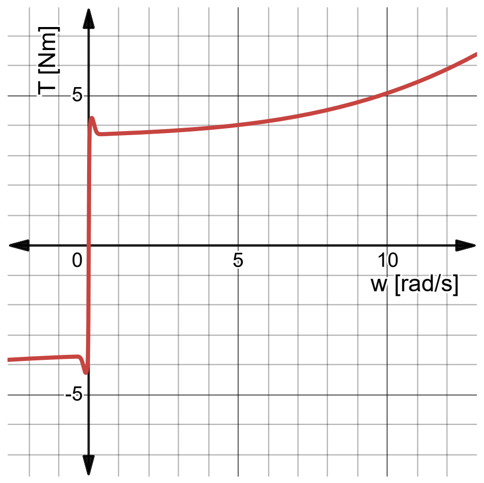

Figure 6 Plotted preview of the total friction torque $\tau_{friction}$ as a function of angular velocity $\dot{\theta}$ using example coefficients for each friction type.

The coefficients used in the preview plot are as follows:

Table 5 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	$[Nm]$
$\dot{\theta}_{brk}$	0.1	$[rad/s]$
τ_{brk}	5.05	$[Nm]$

2.3 Modeling Physical Angular Constraints

Represent the physical angular constraints of the pendulum within the derived model? Or simply note that they exist but will not be modeled directly. But modeled separately in simulation as a stiff spring-damper system at the limits? Or by simply inversing the velocity with a damping factor when the limits are hit?

3 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.

4 Simulation and Results

Here we will simulate the system using MATLAB/Simulink to validate our theoretical findings. The simulation will model the dynamics of the system under various conditions and inputs.

4.1 Angular Constraints

Show that the pendulum has physical angular constraints and discuss how these constraints will be handled in the simulation environment.

5 MCU Implementation

In this section we will attempt to explain and describe the implementation of the designed controller on a MCU. We will cover the hardware and software aspects of the implementation, including the selection of the MCU, interfacing with sensors and actuators, and the programming of the control algorithm. We will also discuss any challenges faced during the implementation process and how they were addressed.

6 Comparison and Discussion

Within this chapter we will compare the performance of the different approaches to the implementation of the designed controllers. We will be comparing the controller performance from simulation, Matlab implementation and MCU implementation. We will analyze the results obtained from each approach and discuss the advantages and disadvantages of each method.

Conclusion

Here we will summarize the key findings of the project, reflect on the challenges encountered during the modeling and controller design process, and suggest potential directions for future work to enhance the system's performance and robustness.

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`.