University of Southern Denmark

Semester Project PRO4

4th Semester Robotics - 2019

Group 04

Control and Regulation of Robotic Systems

Christian Eberhardt	Jakob Grøftehauge	Lars Pedersen
chebe17@student.sdu.dk	jakra17@student.sdu.dk	larpe17@student.sdu.dk

Mads Kuhlmann-Jørgensen	Jakob Møller kaad	Peter Christiansen
makuh17@student.sdu.dk	jakaa17@student.sdu.dk	pechr17@student.sdu.dk

Supervisor: Agus Ismail Hasan

Handover Date: 29-05-2019

Abstract

The purpose of this project is to design and implement control of a pan-tilt system. The system will be controlled through user input in the form of commands sent from a PC. Apart from expertise in the discipline of control theory, the project explores the disciplines embedded programming and digital design. It is required that the control algorithms are implemented in an embedded system and the control signals are sent to the digital system, which acts as an interface to the pan-tilt system. An SPI protocol handles the communication between the two devices. The embedded system comes in the form of a Tiva C-series LaunchPad development kit with a TM4C123GH6PM microcontroller for which programs are written in C and on which a real-time operating system is implemented. The digital system comes in the form of a Basys-3 development kit with an Artix-7 FPGA for which logical expressions are written in VHDL. An UART protocol is developed, that specifies commands available for the user to control the pan-tilt system. The project examines the response of the pan-tilt system using the Ziegler-Nichols and pole placement tuning methods for the PID-controller as well as comparing the response of a single PID-controller and two PID-controllers in cascade. In addition the difference between modelling and implementation is examined by comparing the responses of simulated and implemented PID-controllers.

The project finds that using the cascaded position PID-controller design yields performance worse than that of a single position PID-controller. However this is probably due to poor inner controller design. In addition it is found that with a sufficient model, the pole placement method yields gains suitable for implementation.

Contents

1 Introduction

In the report some words will appear in *italic*. These words will be explained in the word list, section 14. The word will only appear in italic the first time it is presented. Cited material such as datasheets or literature will appear in square parentheses as shown here, [1]. Datasheets, source code, a demonstration video and other material related to the project can be found in the digital appendix. A detailed description of the contents of the digital appendix is to be found in section 13. Flowcharts will be designed using the principles described in the guide found in the digital appendix [2].

2 Problem Description

The theme of the project is regulation and control of robot systems. The project revolves around control of a *pan-tilt system* sketched in figure 1. The pan-tilt system is actuated by two DC motors which are interfaced using H-bridges. The system is equipped with *incremental encoders* on the motors, as well as hall effect sensors to provide *index signals*.

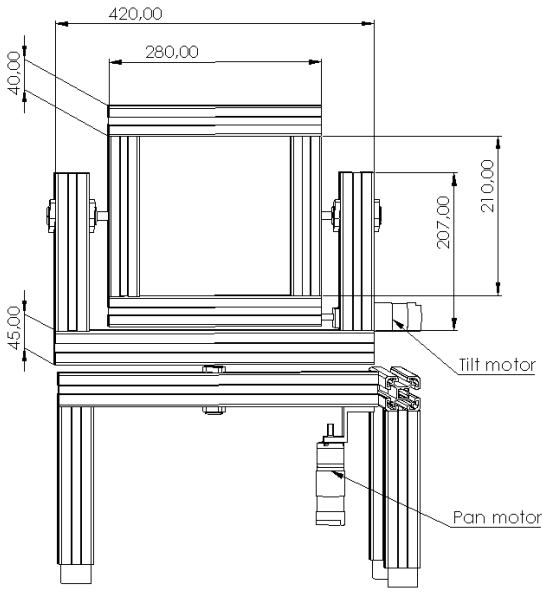


Figure 1: Technical drawing of the pan-tilt system. All dimensions are in mm.

To honour requirements stated in the given project description [1], it has been chosen to set out to design and test several different controller designs and tuning methods to allow control of the angular position of the two motors. This includes developing a mathematical *model* of the system. Reference inputs for controlling the positions must be supplied through a user interface. To accommodate this an embedded program must be designed with appropriate tasks.

Additionally the given project description has the following requirements to the project:

- The control algorithms must be implemented on a microcontroller.
- Data exchanged between the microcontroller and the FPGA must be transmitted using the SPI-protocol.
- The FPGA must control the PWM-signal to the motors through the H-bridges.
- The FPGA must be used for determining the position of the motors using the encoders on the motors.

Based on the requirements stated, figure 2 shows an overview of the system and how the different elements are connected. Furthermore it has been dictated that the pan-tilt system is not to be disassembled.

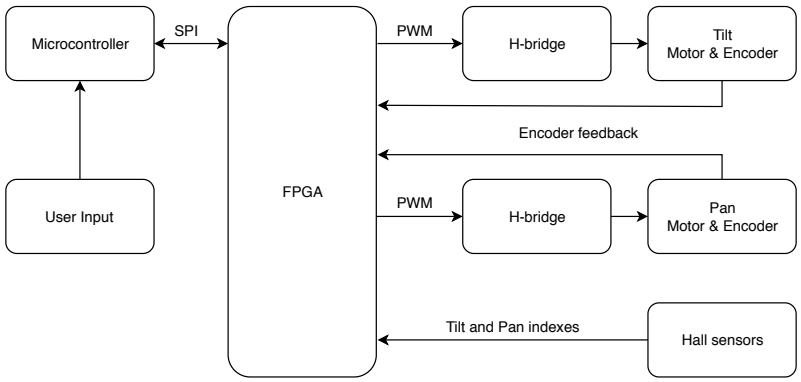


Figure 2: Block diagram of the overall hardware structure.

3 Analysis

This section will provide a description of the elements of the given pan-tilt system as well as an analysis of possible approaches to satisfy the problem description described in section 2.

3.1 Description of Physical Setup

The pan-tilt system is made of aluminium extrusions and can be divided into two sections, a stationary lower section and a upper movable section. The movable section consists of two *frames*, the pan frame and the tilt frame. The tilt frame is a simple square, and the pan frame is u-shaped with the tilt frame attached inside as illustrated on figure 1. The tilt frame can rotate freely while the pan frame is limited in its rotation to around 160° by mechanical limitation.

From initial experiments with the pan-tilt system it is found that significant friction is present in joints and bearings. It is also found that the friction varies according to the position of the frame. Probably these irregularities stem from the physical setup being poorly constructed, as it is visibly skewed.

Each frame is actuated by a 12 V DC motor equipped with an incremental encoder [3]. The resolution of the encoders is 360 counts per revolution of the output shaft. The encoders make it possible to determine an offset from the initial position. The gearing between the motors and the frames on the pan-tilt system is 3:1, so the frame experiences one revolution for every three revolutions the motor experiences. Hall effect sensors provide an index signal for each frame. The index signals are used to provide a reference point for the incremental encoders. From the change in position relative to the time, the velocity of the motor can be calculated. Both motors are connected to an H-bridge, which can be enabled with a PWM-signal to control the voltage over the motors. In accordance with the datasheet of the H-Bridge, L6203 [4], it requires a PWM frequency typically around 30 kHz and up to 100 kHz to function. The H-bridge also makes is possible to change the polarity of the voltage supplied, yielding a change in the motors turning direction.

3.2 Controller Design

Figure 3 shows a simple *controller* setup. The plant is the system which is to be controlled. The controller outputs a control signal which is the input to the plant. In this project the plant would be an electric motor for which the control signal could be a voltage and the output could be an angular velocity. If a perfect model of the plant exists, and no external disturbances act on the system, a controller can be designed that controls the plant purely based on a reference input. This type of system would be classified as open-loop. In reality, such a system cannot be obtained, so to properly control the system, it is beneficial to monitor the output of the plant. The output is subtracted from the reference, yielding an error used as input for the controller. This type of feedback yields a closed-loop system and makes the system more robust.

The problem description states that a position controller has to be designed, however such a controller can take on many forms, leading to different design approaches and methods. Generally speaking, controller design can be divided into two categories, classical control, which operates in the frequency domain, and modern control which operates in the time domain. Both approaches will be examined in this section.

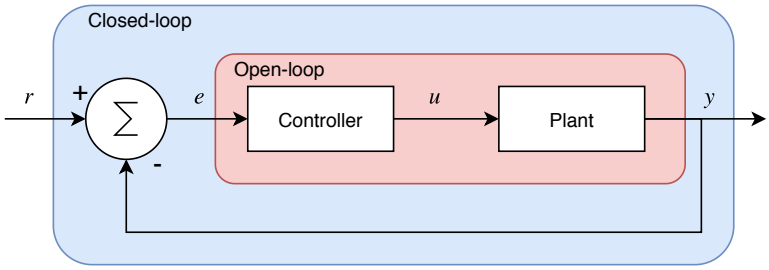


Figure 3: Figure illustrating the concept of controller setup highlighting the open- and closed-loop concepts.

Classical Control Methods

A traditional method for designing a controller for a motor is to use a PID-controller. Though many controller configurations using parts of the PID-controllers design exists, only the full PID-controller will be examined. The block diagram of a PID-controller can be seen on figure 4. As seen on the block diagram a PID-controller utilises a feedback loop. The controller consist of three terms: a proportional term, an integral term and a derivative term. Each term is evaluated separately and then combined to yield the output of the controller. If only the proportional term is used, a steady state error between the output and the reference will always be present. Therefore the integral term is introduced to eliminate the steady state error, as the integral will keep increasing until there is no steady state error. The derivative term adds anticipatory action based on the trend of the error [5].

To achieve a desired response of the system, the different gains of the PID-controller must be found. Finding gains for the PID-controller different methods are available each with different advantages. A method is *pole placement*, where the location of poles in the closed-loop system are strategically placed. This introduces the prerequisite, that an accurate model of the system must be known. Furthermore the designer must know, where to place the poles to achieve the desired response. The root locus provides a method of analysing movement of the poles as an increasing gain is applied to the system. The method makes it possible to tweak the gain until the desired response is achieved. If no model of the plant is available other means for tuning is present. Two heuristic methods known as Ziegler-Nichols methods exist. One is based on creating a marginally stable system, and the other is based on the open-loop step response of the plant. Only the latter will be examined further. This method, assumes that the plant can be described by a first order delayed system.

Physical systems have limitation such as maximum ratings. When modelling such systems these limitations can be taken into account by introducing *saturation* to the control signal. The saturation will level the signal at the saturation limit if the signal exceeds the saturation limits. Having a system in saturation the integral term will continue to windup as long as an error is present between the reference and the output. This is not a desired behaviour, since it will take time for the integral term to decrease for the control signal to go below the saturation limit. To circumvent this, anti-windup can be implemented. When implementing anti-windup it can be executed using back-calculation or conditional integration. The implementation of back calculation can be seen on figure 4. This approach feeds a fraction of the difference between the commanded output signal and the saturated signal back to the integrator. This way keeping the commanded output signal close to the saturation limits, while in saturation. Conditional integration works by feeding zero into the integrator, thus keeping the integration term constant, when the control signal is saturated.

Instead of using a *single controller* for the plant, sometimes it can be beneficial to implement multiple controllers in a *cascaded* design. Here, the plant is perceived as multiple simpler plants that feed into each other. The outermost control loop will then supply the reference signal for the next and so on. If designed correctly, this design will provide better noise rejection and lower sensitivity to parameter variations. This is because disturbances acting on the inner loop will be

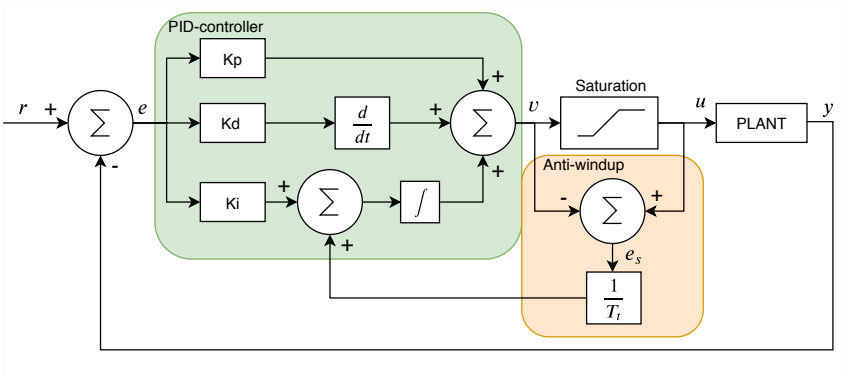


Figure 4: Block diagram of a PID-controller including anti-windup implemented using back calculation. T_t is the tracking time constant and e_s is the saturation error.

isolated from, and ideally invisible to the outer control loops. On the pan-tilt system a cascade constituting three controllers would be a inner current controller, a velocity controller and then an outer position controller.

Modern Control Methods

A system can be described based on the dynamics of the states found in the system. For the DC motor example, the states could be current, angular velocity and angular position. The dynamics can be presented using the state space form shown in equation 1,

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \quad , \end{aligned} \tag{1}$$

where A is the system matrix, B is the input matrix, C is the output matrix, x is a vector containing the states and u is a vector containing external control signals. Modern control aims to control the states in the system via state feedback, which introduces the control law seen in equation 2,

$$u = Fx \quad , \tag{2}$$

where F is the feedback matrix, which can be designed to yield the desired system response.

One approach to design the feedback matrix is pole assignment, where the feedback matrix is calculated to yield specific closed- loop poles in order to achieve a desired response. This requires the designer to decide exactly where the poles should be placed, which for somewhat complex systems can be difficult. A Different approach is using the linear quadratic regulator method. This algorithm is based around the minimising of a cost function concerning the optimisation of operating a linear system. This method provides a way to apply a weight to each state and each control signal and thereby influence which will be prioritised higher. As an example, the states can be prioritised highly giving a fast response, at the cost of more aggressive control signals.

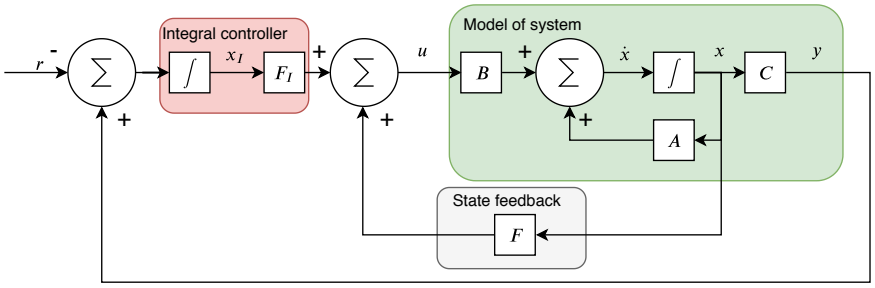


Figure 5: Block diagram of an integral controller.

Since both of the these design methods are based on a linear model of the system, it is a prerequisite that a very accurate model of the plant, which is defined by the A and B matrices, is known.

A way to reduce some of the issues from uncertainties about the plant, is to include integral action as seen on figure 5. The integral feedback provides zero steady state error by integrating the error. Taking basis in the state space form equation 1 and adding an additional feedback, the input is given as described in equation 3,

$$u = Fx + F_I x_I \quad , \tag{3}$$

where F is the state feedback matrix, F_I is the integral matrix and X_I is described in equation 4,

$$x_I(t) = \int_0^t y(\tau) - r(\tau)\, d\tau \quad , \tag{4}$$

where t is the time, y is the output, r is the reference and τ is the integration variable.

State feedback requires that every state is measured, which is rarely possible. However means to estimate states of a given system are available. The estimation can be achieved by introducing an observer to the system, which is illustrated on figure 6. The observer aims to mirror the plant and by that making it possible to approximate a given state. The observer introduces an observer gain, L, which is proportional with the difference between the output from the plant and the estimated observer output. The observer state space model is as seen in equation 5,

$$\begin{aligned} \dot{\hat{x}} &= A\hat{x} + Bu + L(C\hat{x} - y) \\ \hat{y} &= C\hat{x} \quad , \end{aligned} \tag{5}$$

where $\hat{x}$ is the estimated state and $\hat{y}$ is the estimated output.

As with the classical control, saturation might be considered in modern control. This can be handled by introducing a matrix M, as seen in figure 6, which can alter the system dynamics when in saturation. This can act as a form of anti-windup.

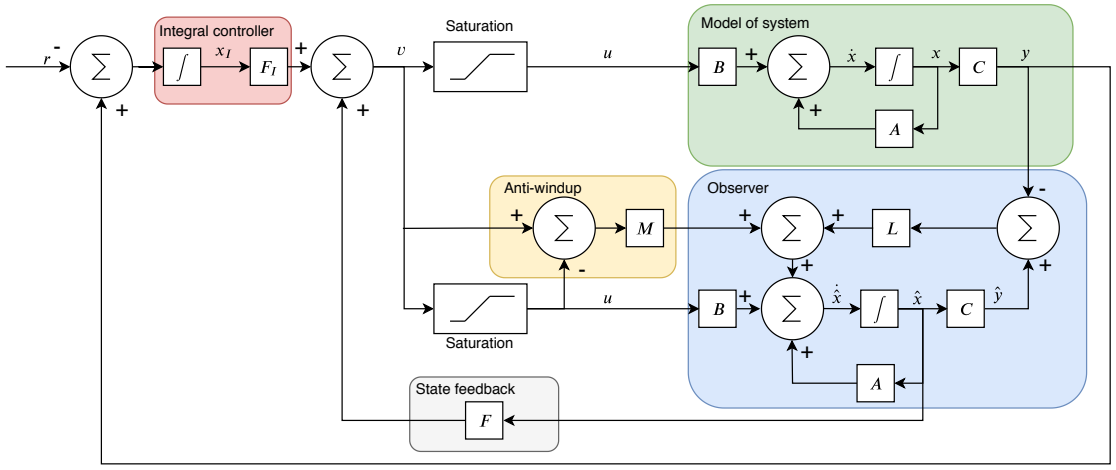


Figure 6: Block diagram of an integral controller with observer and anti-windup.

3.3 Implementation of Discrete Controller

Upon designing a controller it is important to keep in mind that implementation in software is discrete rather than continuous. The plant, for which the controller is designed, is continuous. Since the plant and the controller exists in different domains, mathematical analysis of the combined system is not possible. One way to get around this, is to design the controller in continuous time while effects of the discretisation are kept in mind. The controller can also be designed directly in the discrete domain. However, that requires the plant model to be converted into a discrete time representation. Different methods can be chosen when discretising a continuous system. These are based on different approaches to numerical integration. Three common approaches exist, the forward rectangular rule, the backward rectangular rule and the trapezoidal rule. Each has advantages and disadvantages, although only the trapezoidal method will be examined further, as it gives the best approximation of the integral.

Implementing a discrete PID-controller, utilising the Tustin method [6], can be done as seen in equation 6,

$$\begin{aligned} u(kT+T) = {} & k_P \cdot e(kT+T) \\ & + u_I(kT) + k_I \frac{T}{2} \cdot (e(kT+T) + e(kT)) \\ & + k_D \frac{2}{T} \cdot (e(kT+T) - e(kT)) - u_D(kT) \quad , \end{aligned} \tag{6}$$

where $u(kT+T)$ is the control signal in discrete time. T is the sampling time, e is the error and k_p, k_I and k_D are the PID gains. Tustin's method is based on the trapezoidial rule, and dictates a certain way of implementing the integral and thereby also the differential part. As seen from equation 6 the terms depend on the sampling time, thus it needs to be known. A rule of thump states that the sampling frequency for a discrete controller, should be approximately 20-30 times larger than the closed-loop bandwidth of the system [7].

3.4 Microcontroller

Implementing at least two controllers on the same microcontroller and having to accommodate other tasks in parallel, it would be advantageous to implement some sort of scheduling algorithm. The easiest way of implementing a scheduling algorithm is by the use of an operating system. Various types of operating systems exist, though in this project only two operating systems have been evaluated, Run To Complete Scheduler (RTCS) seen in section 13 and FreeRTOS. The RTCS uses a cooperative scheduling algorithm, where the task voluntarily has to yield processing time on the CPU. This method provides little overhead, though the risk of a task blocking the CPU, is greater as no master can reallocate resources while another task in running. As default, FreeRTOS uses a preemptive scheduling algorithm, where a central master is responsible for the allocation of resources. The central master can preempt a task whenever it is necessary, therefore minimising the probability of starvation occurring. FreeRTOS, using preemptive scheduling, does however provide a greater overhead compared to the RTCS, using cooperative scheduling. If time-critical tasks exists in the system, it is preferred to use an operating system providing *real-time* capabilities. A real-time system ensures that important tasks are guaranteed to complete within a given time frame from when the task is signaled to run. FreeRTOS provides functionalities to operate as a real-time system, such as prioritisation of tasks.

4 Delimitation

As described in section 3, there are a broad range of solutions regarding controlling the system. Some of the solutions might be equally valid, but offer different approaches to the project. This section concerns the choices made in the project and will found the basis for the requirement specifications.

4.1 Physical Environment of the Project

As mentioned in section 2 some requirements from the project description has to be met. The controllers will be implemented on the *TM4C123GH6PM* microcontroller as part of the Tiva C-series LaunchPad development kit and the motor interface will be implemented on the *Artix-7* FPGA on the Basys-3 development kit. This is due to experience from lectures concurrent with the project. While it is recognised that other and better solutions might be available, it has not been of priority to find these due to the scope of the project.

4.2 Controller

As mentioned in section 3, different types of controller methods are available. The modern control methods using state feedback offers a rather complex solution, where a precise model of the system is crucial to achieve the wanted behaviour of the controller. If the model of the system does not

comply with the physical system, the controller will be subject to a lot of tuning and tweaking. With a vast number of tuning parameters the tuning can be difficult. A classic control method such as the PID-controller offers a simpler design with fewer parameters to tune. Thus it is chosen to work towards implementing PID-controllers. When tuning the controllers it is desired to examine different methods of tuning and comparing these. It is desired to test the pole-placement method and the Ziegler-Nichols step response method. Likewise it is wanted to examine the behaviour of the system using different constellations of the controllers. Thus it is worked towards implementing a single controller as well as cascaded controller, for the pan and tilt motors, on the microcontroller. The controllers will be designed in continuous time, after which, they transformed into discrete time using Tustin's method, for implementation on the microcontroller.

Initially all three states of the system was to be measured, namely the position, the velocity and the current. The position and velocity can be deduced from the encoders, as described in section 3. The current is a high frequency signal making it difficult to measure and control using available hardware. For this reason measuring the current was discontinued. Thus position and velocity controllers are to be implemented.

It is assumed when increasing the sampling rate, that performance of a discrete controller only improves. Therefore it has been decided to run single controllers, as well as inner loops of a cascaded design, at a rate of 1000 Hz. The outer controller of a cascaded design will run at 200 Hz. It has later been discovered that this rate may be excessive, which will be discussed in section 10.

A priority of the project will be to examine the difference between the simulation using the model and the implementation on the physical system. In conclusion the focus of the project is to examine different methods and approaches of controlling the system rather than work towards a specific application. A reference for the controllers transmitted via UART from the computer to the microcontroller is deemed sufficient as user input. The UART protocol is chosen due to previous experience with the protocol, making it an obvious choice.

4.3 Embedded System

It is chosen to use FreeRTOS v10.2.1 as operating system for the microcontroller, because it offers key features for implementing multiple tasks to run in parallel. Furthermore FreeRTOS introduces prioritisation of task without compromising the timing of individual tasks when having multiple time critical tasks. Beyond that FreeRTOS provides functionality for queues, mutex' and semaphores. When using the preemptive FreeRTOS, it must be ensured that the *overhead* is not an issue.

In the project description section 2 it is stated that an SPI connection has to be used for exchange of data between the FPGA and the microcontroller. Therefore hardware *modules* supporting the SPI protocol has to be implemented on the FPGA. Furthermore the FPGA must be able to interface the *motor encoder* keeping track of the position and the velocity. The FPGA also has to produce the PWM-signal controlling the motor, the frequency of the signal has to comply with the specification of the H-bridge introduced in section 3. For regulating the pan-tilt system to

a desired position it is essential to have a reference position. To be able to obtain the reference position, reset functionality is to be build into the FPGA, to ensure that the pan-tilt system will return to a defined zero position.

5 Requirement Specifications

In accordance with section 2 and section 4, different PID-controllers are to be designed and the following must be examined:

- The effect of different tuning methods for the controller, namely the Ziegler-Nichols and pole placement method.
- The effect of using a single controller design versus a cascaded controller design with a velocity controller in the inner loop and a position controller in the outer loop.

The following specifications shall be used as guidelines when designing the PID-controllers:

- The controllers are to have a maximum rise time of 0.5 s, a maximum overshoot of 5 % and a maximum 1 % settling time of 1.5 s.

To meet the aforementioned goals, a program which allows for processing PID-controllers must be implemented on the microcontroller. To ensure reliable performance, the following requirements must be met:

- The PID-controllers must be implemented on the microcontroller.
  - The PID-controllers must not utilise more than 60 % of the CPU time on the microcontroller.
  - A maximum execution time for the PID-controller must be determined.
- The microcontroller must run the FreeRTOS operating system.
  - It must be verified that the overhead does not compromise the timing of the system.
- A user interface protocol is to be implemented to allow for user input through UART.

Using an FPGA the motor encoders must be interfaced to provide feedback for the PID-controllers executing on the microcontroller. The interface must meet the following requirements:

- Data transmission between the FPGA and the microcontroller must be possible through SPI.
- Using SPI with a bit rate of 13.3 Mbit, the FPGA is to receive 99.9 % messages correct with no errors.

- The FPGA is to be able to keep track of the position and velocity of the motors via the encoders.
- Provided with a reference from the microcontroller, the FPGA must be able to generate an appropriate PWM signal with a frequency between 30 kHz and 100 kHz.
- A reset function for the pan-tilt system is to be implemented to allow for determining a reference position.

6 System Analysis and Modelling

In order to analyse and the pan-tilt system and design appropriate controllers, a mathematical model describing the dynamics of a DC motor has to be derived. First the electric equivalent circuit of the DC motor is analysed. As seen in figure 7 the motor can be described as a voltage source in series with a resistor, R_m, an inductor, L_m, and another voltage source representing the *back emf*. To setup the system equation it is assumed that the torque is proportional to the angular velocity of the motor shaft. Furthermore, it is assumed that the motor torque is proportional to the current, i. If the motor torque and the back emf constant are represented in SI units, they are equal to each other [8]. This constant will henceforward be denoted as one constant, K_m. The system equations is derived using figure 7 as well as Newtons's second law and Kirchoff's voltage law. The equations can be seen in equation 7 and 8.

$$J\ddot{\theta} + b\theta = K_m i \tag{7}$$

$$L_m \frac{di}{dt} + R_m i = V - K_m \dot{\theta} \tag{8}$$

Using equation 7 and 8 the system is represented on state space form seen in equation 9.

$$\begin{aligned} \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{i} \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & \frac{-b}{J} & \frac{K_m}{J} \\ 0 & \frac{-K_m}{L_m} & \frac{-R_m}{L_m} \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{L_m} \end{bmatrix} \\ y &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ i \end{bmatrix} \end{aligned} \tag{9}$$

The system dynamics can also be described by the transfer functions for velocity and position, seen in equation 11 and 10.

$$\frac{\dot{\theta}(s)}{V(s)} = \frac{K_m}{(J \cdot s + b)(L_m \cdot s + R_m) + K_m^2} \tag{10}$$

$$\frac{\theta(s)}{V(s)} = \frac{K_m}{s \cdot ((J \cdot s + b)(L_m \cdot s + R_m) + K_m^2)} \tag{11}$$

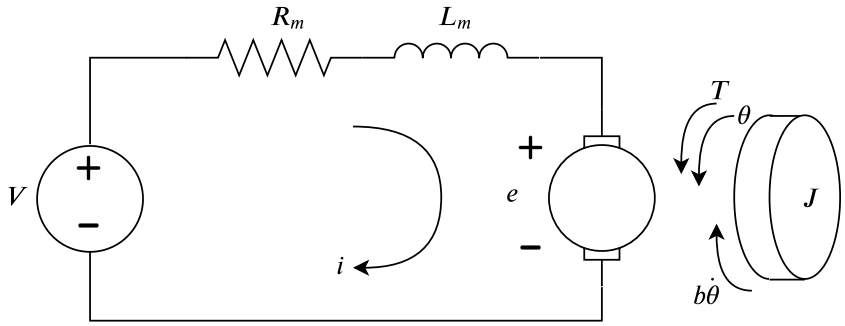


Figure 7: Electrical equivalent circuit diagram of a DC motor and a free-body diagram of the motor shaft. T represents the motor torque, J the inertia of the motor, θ the position of the shaft and $b\dot{\theta}$ the friction torque [8].

6.1 Determining System Parameters

Moment of Inertia

An estimation for the value of the moment of inertia will be calculated based on measurements and information from data sheets. This section will summarise the results of the calculations, the full derivations can be found in the document "Derivations and Estimations for Inertia" [9].

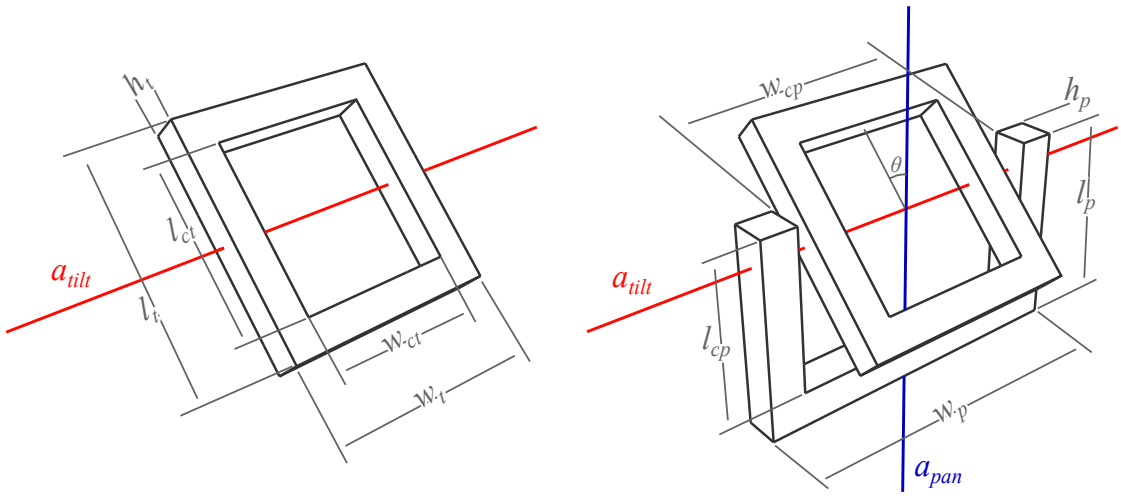
The inertia estimates are based on a simplified representation of the pan-tilt system shown in figure 8. This representation ignores bolts, brackets, gears, etc. Furthermore, the extrusions are considered as solid boxes, thus ignoring the profile shape.

Based on the simplified representation, the inertia of the tilt frame can be found by subtracting the inertia of the inner box of dimensions $w_{ct} \times l_{ct} \times h_t$ from the inertia of the outer box with dimensions $w_t \times l_t \times h_t$. This yields equation 12,

$$J_{a_{tilt}} = w_t\, h_t\, l_t\, \rho_t \frac{1}{12}(l_t^2 + h_t^2) - w_{ct}\, h_t\, l_{ct}\, \rho_t \frac{1}{12}(h_t^2 + l_{ct}^2) \quad , \tag{12}$$

where ρ_t is the average density of the tilt frame.

The inertia of the pan frame can be found in a similar manner, however to properly estimate the inertia relevant to the pan motor, the tilt frame, must be considered as well. The inertia of the tilt frame, about the axis a_{pan}, depends on the angle, θ, at which the tilt motor is positioned. Finding the inertia of the inclined tilt frame is non-trivial and requires triple integration. The resulting formula is non-linear as it will give the largest value at $\theta = \frac{\pi}{2}$ and the lowest at $\theta = 0$. It has been chosen to calculate the inertia at $\theta = \frac{\pi}{4}$, to get a value which is right between the two extremes.



(a) Representation of the tilt frame.

(b) Representation of the pan and tilt frame.

Figure 8: Simplified representation of the pan and tilt frames used for estimating the inertia.

The total inertia relevant to the pan motor, $J_{a_{pan}}$, can then be found in equation 13,

$$\begin{aligned} J_{p,\,a_{pan}} &= w_p\, l_p\, h_p\, \rho_p \frac{1}{12}(h_p^2 + w_p^2) - w_{cp}\, l_{cp}\, h_p\, \rho_p \frac{1}{12}(h_p^2 + w_{cp}^2) \\ J_{t,\,a_{pan}} &= \frac{1}{24} h_t\, l_t\, w_t\, \rho_t (h_t^2 + l_t^2 + 2w_t^2) - \frac{1}{24} h_t\, l_{ct}\, w_{ct}\, \rho_t (h_t^2 + l_{ct}^2 + 2w_{ct}^2) \\ J_{a_{pan}} &= J_{p\,a_{pan}} + J_{t\,a_{pan}} \quad , \end{aligned} \tag{13}$$

where $J_{p\,a_{pan}}$ is the inertia of the pan frame and $J_{t\,a_{pan}}$ is the inertia of the tilt frame, both about the pan axis a_{pan}.

The densities $\rho_t = 987.7\,\mathrm{kg/m^3}$ and $\rho_p = 996.3\,\mathrm{kg/m^3}$ are based on values found from technical data of similar aluminium extrusions [10] [11]. Evaluating using these values, and the measurements in figure 1 it is found that $J_{a_{pan}} = 0.0602\,\mathrm{kg\,m^2}$ and $J_{a_{tilt}} = 0.0169\,\mathrm{kg\,m^2}$. Since the model is based on the angular motion for the motor shaft, and not the angular motion of the pan and tilt frames, these values will have to be divided by three. This is because of the gear ratios which dictate that the pan and tilt frames have one third the angular velocity of the motor shaft for the respective frames.

Motor Coefficients

The coefficients of the tilt motor are found experimentally, with the approach described in the Danish article "Modeldannelse" [12]. Same parameters are assumed for the pan motor. In this approach, the angular velocity and current is measured for different voltages. Equation 14 can be used to describe the relationship between the angular velocity, current and voltage, the armature

voltage, u_a, and the external load, τ_b for the motor:

$$\begin{aligned} \omega_1 K + i_{a1} R_a &= u_{a1} \\ i_{a1} K - \tau_c - \omega_1 b &= \tau_{b1} \\ \omega_2 K + i_{a2} R_a &= u_{a2} \\ i_{a2} K - \tau_c - \omega_2 b &= \tau_{b2} \end{aligned} \quad , \tag{14}$$

where K is a coefficient related to the torque of the motor, τ_c is the static friction, b is the dynamic friction in the system and R_a is the internal resistance in the motor windings. $\tau_{b1} = \tau_{b2} = 0$ when the motor is not operating with a load and the voltage is varied. ω_1 and i_{a1} are the angular velocity and current respectively at the voltage u_{a1}, while ω_2 and i_{a2} are a different set of angular velocity and current measurements at the voltage u_{a2}. For the experiment 7 different sets of angular velocity, voltage and current measurements are taken and a linear expression is fitted to the results. From the linear expression two sets of values for voltage, current and angular velocity is extracted for the computation of the motor coefficients. Further results and calculations can be seen in the digital appendix outlined in section 13. The parameter values found can be seen in table 1. $\tau_c = 0.139\,\mathrm{N}$ is not included as the static friction is ignored for the dynamic model.

6.2 Analysis of the Modelled Plant

Through discussion with the project supervisor, it has been decided that determining the inductance L_m is not of interest. Instead the value found in the control tutorial [8], provided by the supervisor, is used.

The state space model in equation 9 combined with the model parameters from table 1 is used to calculate the step response and pole locations for the plant seen in figures 9 and 10.

Parameter		**Pan**	**Tilt**
Moment of inertia	J [kg m^2]	0.02	$5.6 \cdot 10^{-3}$
Inductance	L_m [H]	$2.75 \cdot 10^{-6}$	$2.75 \cdot 10^{-6}$
Resistance	R_m [Ω]	4.65	4.65
Back emf/motor torque constant	K_m [V s/rad]	0.49	0.49
Viscous friction constant	b [N m s]	$7.38 \cdot 10^{-4}$	$7.38 \cdot 10^{-4}$

Table 1: Parameters used for modelling the pan and tilt motor. R_m, K_m, and b is found experimentally, J is estimated and L_m is from [8].

Analysing the location of the poles, it is apparent that each plant consist of one fast pole and two slower poles. This also complies well with the fact that the plant is comprised of an electrical part and a mechanical part. The electrical part is by nature significantly faster than the mechanical part, therefore yielding the fastest pole. For the dynamics of the system these fast poles are negligible, as the slower poles are dominant. Looking at the zoomed in plot in figure 9 it can also be seen that the difference in inertia between the two plants manipulates the second fastest pole. Since the tilt frame has a lower moment of inertia, it can be seen that the system becomes faster as the moment of inertia decreases.

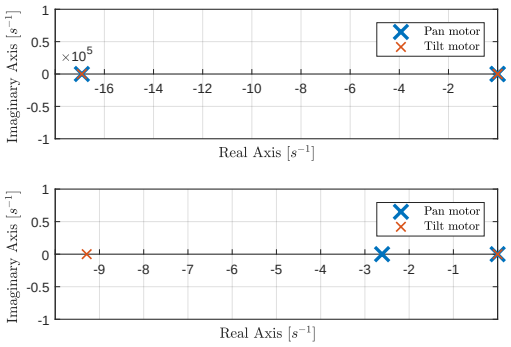


Figure 9: Pole Locations for the motor models. Bottom plot is zoomed to show the slower poles.

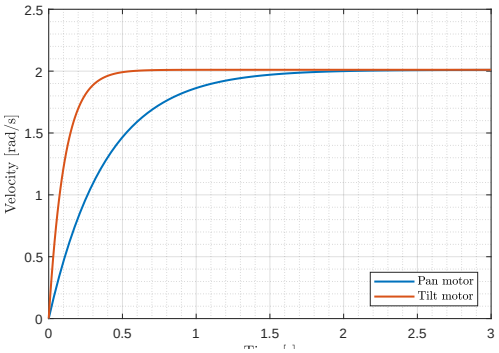


Figure 10: Velocity step response for the motor models.

The observations based on the pole locations can be verified by looking at the step response for the velocity of the systems. From figure 10 it is seen that the velocity responses resemble that of a first order system, although the transfer function for the velocity would be of second order. Thus it is verified that the fast pole for each plant is negligible. It can also be seen that the response of the tilt motor is significantly faster.

6.3 Conclusion

A mathematical model of the dynamics of the DC-motors in the pan-tilt system has been derived. The moment of inertia has been estimated from equations, which describe the moment of inertia for shapes similar to the ones found in the pan-tilt system. The coefficients describing the behaviour of the DC-motor has been determined experimentally by evaluating related measurements of current, voltage and angular velocity. From the mathematical model the system is put on state space form in order to calculate step responses and analyse the location of the poles.

7 Control System Design

As mentioned in section 5 the controller design method explored in this project will focus on different configurations of a PID-controller. Both single and cascaded controller will be examined, meaning that both position and velocity controllers are to be designed. The performance specifications mentioned in section 5 is used as a guideline of the design. However, the main focus will be on comparing different controller configurations rather than meeting the performance specifications. As mentioned in section 4, the controllers designed in this section will be designed in the continuous time domain, afterwards additional steps will be taken to approximate the effect of the discrete controllers and verify performance.

7.1 Design of Position Control Based on Pole Placement

A single position PID-controller must be designed for each motor. To design the control systems, the root locus plot will be utilised to calculate the appropriate gains for the proportional, integral and derivative terms. Considering a closed-loop system of the form shown in equation 15,

$$H_{cl}(s) = \frac{K \cdot G(s)}{1 + K \cdot G(s)} \quad , \tag{15}$$

with a proportional controller K and plant $G(s)$, the root locus shows how the closed-loop poles move as the gain of the controller changes. With gain approaching infinity, the closed-loop poles will move either towards the open-loop zeros or infinity. By designing a controller to strategically place zeros, and choosing an appropriate gain, the position of the closed-loop poles can be controlled.

As seen from the transfer function of a PID-controller in equation 16,

$$K(s) = \frac{k_d s^2 + k_p s + k_i}{s} \quad , \tag{16}$$

where k_p,k_d and k_i are the respective gains, this type of controller adds two open-loop zeros and one open-loop pole to an existing system. The open-loop pole, originating from the integral term, will always be placed at the origin, but the positions of the zeros can be controlled. The design procedure which will be followed in this section will thus involve first placing these two zeros and afterwards choosing an appropriate gain based on the root locus plot. To place the zeros, equation 17 is used,

$$K(s) = \frac{(s - z_1) \cdot (s - z_2)}{s} \quad , \tag{17}$$

where z_1 and z_2 describe zero positions.

To figure out where to place the zeros, it must first be considered where the poles for the closed-loop system should ideally be placed. In order to achieve the predominant goal of having a stable system, all closed-loop poles must be placed to the left of the imaginary axis in the complex plane. To further determine appropriate locations for the poles, the time domain specifications described in section 5 must be considered. These can be used to produce frequency domain rules, which indicate an area in the complex plane in which the poles should be located to honour the specifications. These rules are based on the transfer function of a second order system without zeros shown in equation 18,

$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \quad , \tag{18}$$

where ω_n is the undamped natural frequency of the system and ζ is the damping ratio. The closed-loop system for position control has four poles, although only three dominant ones, and two zeros,

so the frequency domains rules will have to be used only as guidelines. To obtain a rise time shorter than t_r the minimum magnitude, w_n, of the poles can be approximated as $w_n \geq \frac{1.8}{t_r}$. To obtain a 1 % settling time faster than t_s, the magnitude of the real part of the poles, $\sigma = \omega_n \zeta$ can be found as $\sigma \geq \frac{-\ln 0.01}{t_s}$. Lastly, to keep an overshoot below 5 %, it must be true that $\zeta \geq \sqrt{\frac{(\frac{\ln 0.05}{-\pi})^2}{1+(\frac{\ln 0.05}{-\pi})^2}}$, which defines a cone shaped region in the complex plane, in which the poles should be placed [13]. The region based on the specifications can be seen on figure 11.

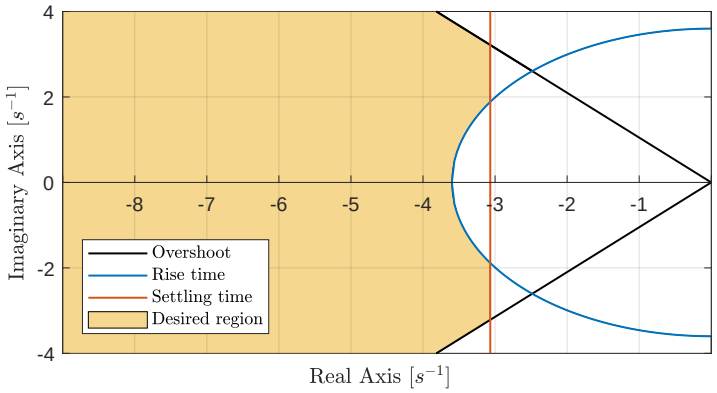


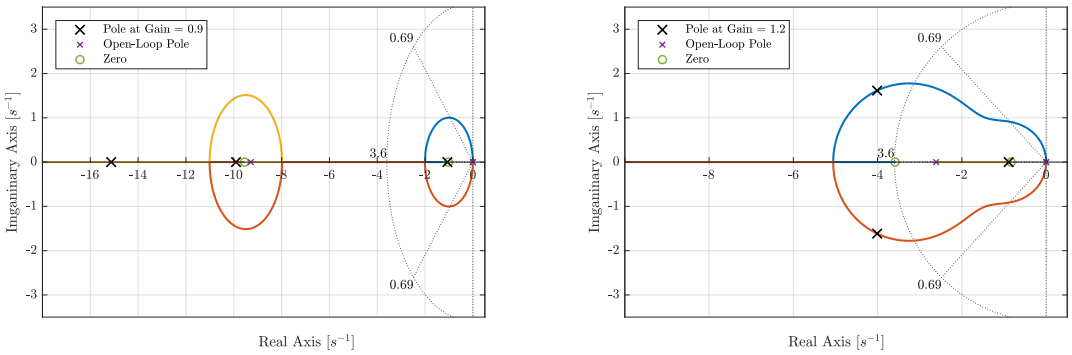
Figure 11: All poles should be placed in the yellow area for the system to meet the required performance specifications.

It can be seen on figure 12 that in order to follow the frequency domain guidelines found above, the two open-loop poles at the origin should be moved to the left. This should be done while keeping their imaginary parts close enough to zero to stay within the cone shaped region dictated by the requirement of 5 % overshoot. Through experimentation it has been found that placing zeros at $s = -9.5$ and $s = -1$ for the tilt motor and at $s = -3.3$ and $s = -0.8$ for the pan motor, produced satisfactory root loci, which are shown in figure 12. Based on the plots, an appropriate gain is chosen. For the tilt motor a gain of $s = -0.9$ was chosen while $s = -1.2$ was chosen for the pan motor. The resulting control gains are summarised in table 2. The closed-loop pole locations at these gains are also visible from figure 12.

Design methods	k_p	k_i	k_d
Position tilt pole placement based	9.45	8.55	0.9
Position pan pole placement based	4.92	3.17	1.2
Velocity tilt pole placement based	1.42	7.80	0.065
Velocity tilt Ziegler-Nichols	0.52	22.86	0.003
Cascade pole placement based	7.53	7.59	1.03
Cascade Ziegler-Nichols	8.78	7.64	1.22

Table 2: Controller gains found from the different design methods.

As stated in section 3, the sampling frequency shall be 20-30 times the bandwidth of the closed-loop



(a) Root locus pan motor.

(b) Root locus tilt motor.

Figure 12: Root locus plots of the pan and tilt system with the chosen zero locations.

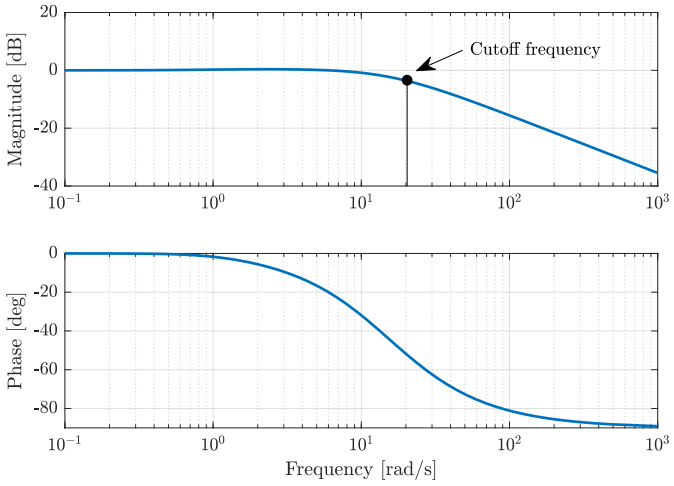


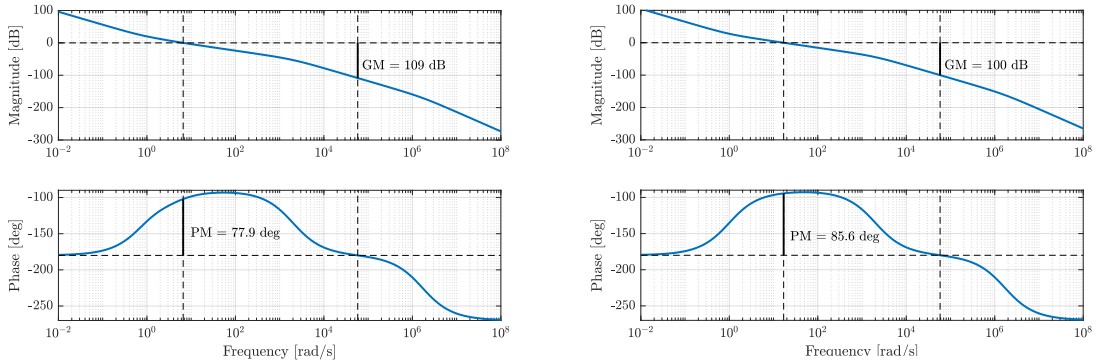
Figure 13: Bode plot of the closed-loop control system for the tilt motor.

system. A bode plot of the closed-loop system for the tilt motor, shown in figure 13 reveals a cutoff frequency of 18 rad/s. The tilt motor bode plot is highlighted since it has the highest bandwidth. A similar plot for the pan motor can be found in the digital appendix, outlined in section 13. With a sampling frequency of 1000 Hz, the position controller honours the minimum sampling rate of $t_{sampling} = \frac{18}{2\pi} \cdot 30 = 85.94\,\text{Hz}$. To emulate the discrete controller, when evaluating performance, a delay of half of the sampling time is added to the controller. This is done by appending the transfer function shown in equation 19,

$$G_{delay}(s) = \frac{1}{\frac{t_{sampling}}{2} \cdot s + 1} \quad , \tag{19}$$

where $t_{sampling} = 0.001\,\text{s}$, to the transfer function of the PID-controller. With this delay added, the phase and gain margins are evaluated by drawing the bode plot of the open-loop transfer function and evaluating the phase at the frequency. This yields 0 dB gain, and evaluating the gain at the frequency which gives $-180°$ phase. As seen in figure 14, the gain margins for the tilt and pan motors are 100 dB and 105 dB respectively, while the phase margins are 85.6° and 77.9°. These

margins are quite large, which is good, since that allows room for more uncertainties in the plant model. Future bode plots and margin plots referenced in this section can be found in the digital appendix.

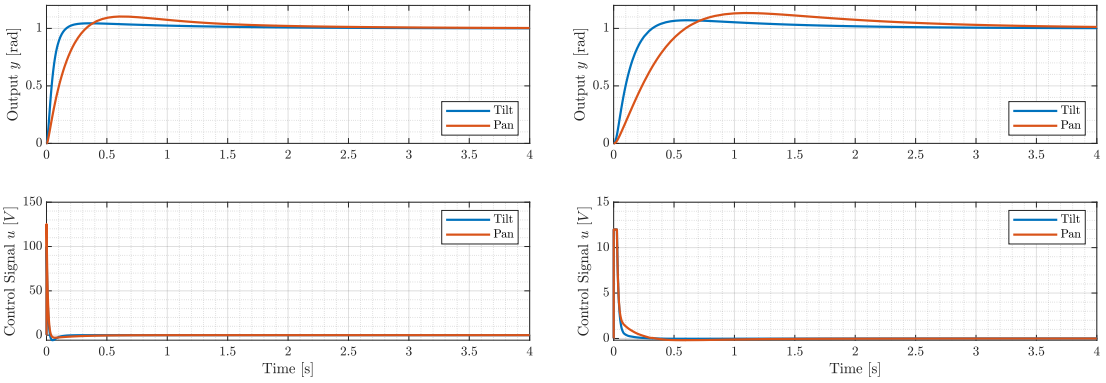


(a) Pan motor.

(b) Tilt motor.

Figure 14: Plots of the phase and gain margins motor control systems for the pan and tilt motors.

The tilt and pan motor systems with added PID-controllers, are simulated using a step reference of magnitude 1. This is done to verify the performance of the controllers. To properly emulate the physical system, saturation and associated anti-windup must be added as described in section 3.2, since the physical system can only respond to a limited control signal. Figure 15a and 15b show the difference that the saturation makes. The figure shows that ideally, the controller for the tilt motor honours the specifications, however the specifications for settling time and overshoot are not quite met when saturation is introduced. Even without the saturation, the step response of the pan motor controller does not quite live up to the requirements, but through experimenting with different zero placement and gains, it was found to be the best compromise between a relatively fast rise time and settling time, while still keeping the overshoot reasonable.



(a) Without saturation.

(b) With saturation.

Figure 15: Step responses of the designed control systems with and without saturation taken into account.

7.2 Velocity Controller Design

As mentioned in section 4, it is wanted to explore the performance of a cascaded controller design. To achieve this, a velocity controller must be designed. As mentioned in section 3 the range of motion for the pan motor is physically limited. Therefore it has been prioritised to focus on designing a velocity controller for the tilt motor, as the unlimited range makes testing the velocity controller possible, without jeopardising the pan-tilt system. Two different methods of designing velocity controllers will be explored. One will use the pole placement approach used in section 7.1, the other will utilise the Ziegler-Nichols approach, which can be used without a mathematical model of the system to be controlled. When working with a cascade control design, it is desired that any inner controllers are faster than the outer loop, as well as critically or over damped [14]. It is therefore worked towards designing a controller yielding no overshoot while still achieving a fast rise time.

Ziegler-Nichols Tuning

The Ziegler-Nichols method uses the step response of the physical system to calculate the controller gains. When using the Ziegler-Nichols method it is not possible to dictate performance specification. Instead, the method aims to achieve a decay ratio, ζ, of approximately 0.25, for the closed-loop system. The method assumes that the plant can be described by a delayed first order system of the form seen in equation 20,

$$\frac{y(s)}{u(s)} = \frac{A_{ss}}{\tau s + 1} e^{-st_d} \quad , \tag{20}$$

where A_{ss} is a steady-state magnitude, τ is a time constant and t_d is time delay. To calculate the controller gains a set of equations are provided. The equations differ depending on the controller type. The equations used for calculating the gains for a PID-controller can be seen in equations 21,

$$k_p = \frac{1.2}{RL} \ , \quad k_i = \frac{k_p}{2L} \ , \quad k_d = \frac{k_p}{0.5L} \quad , \tag{21}$$

where the constants R and L are identified from the step response. R is found by drawing magnitude the tangent to the steepest part of the transition between zero and the steady state. The gradient of the tangent is equal to R, found by $R = \frac{A_{ss}}{\tau}$. L is equal to the latency of the system, and is defined as the distance between the y-axis and the intersection between the tangent and the x-axis [15].

In figure 16 the data points sampled from the velocity step response of the physical system are plotted. The initial spike marked with the yellow ellipse the figure might be caused by friction but will not be discussed further. Due to the seemingly noisy data in the plot, the Curve Fitting tool in MATLAB has been used to fit a curve that, it is displayed as the red line. The red line is described as the expression introduced in equation 20. Using MATLAB $R = 203.4$ and $L = 11.4 \cdot 10^{-3}$

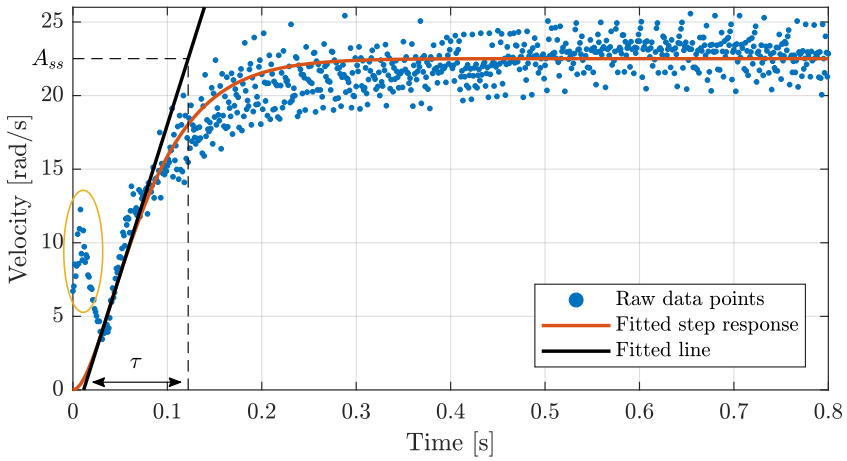
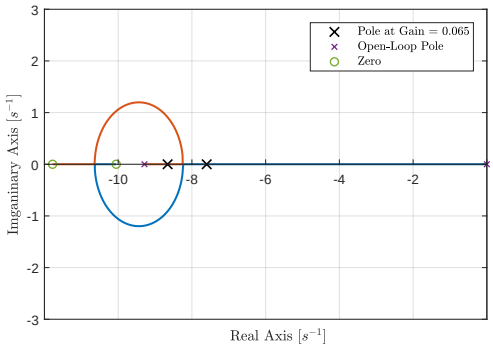


Figure 16: Open-loop velocity step response for the tilt motor, used for tuning using Ziegler-Nichols.

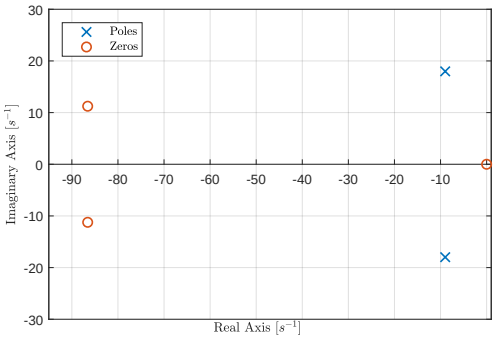
are found. Using the found values for R and L and the equations introduced in equation 21 the controller gains are found and can be seen in table 2.

Pole Placement Based Velocity Controller Design

The procedure for designing the velocity controller based on the model is very similar to the approach used for designing the position controllers mentioned in section 7.1. The main difference is that the open-loop system now has one less pole at the origin, as the only pole at the origin originates from the integral term of the PID-controller. As seen on figure 17a, it has been found that placing the zeros stemming from the PID-controller at $s = -10$ and $s = -12$, yields adequate root loci. This placement allows the close-loop poles to be moved far into the left half plane, while still remaining directly on the real axis. A gain value of 0.065 is chosen. The closed-loop pole locations are visible from the figure 17a. The closed-loop pole and zero locations can be compared to the pole and zero locations given by the Ziegler-Nichols tuning shown in figure 17b.



(a) Root locus plot for the tilt velocity controller with the chosen gains from pole placement.

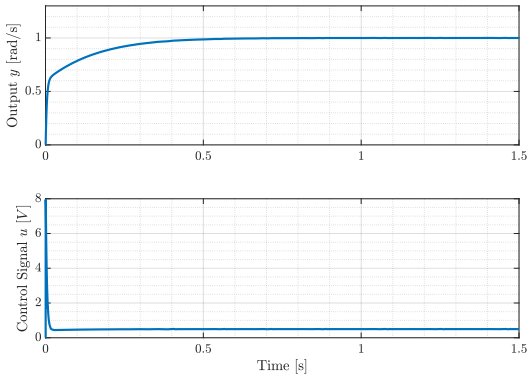


(b) Closed-loop poles and zero locations obtained using the Ziegler-Nichols tuning method.

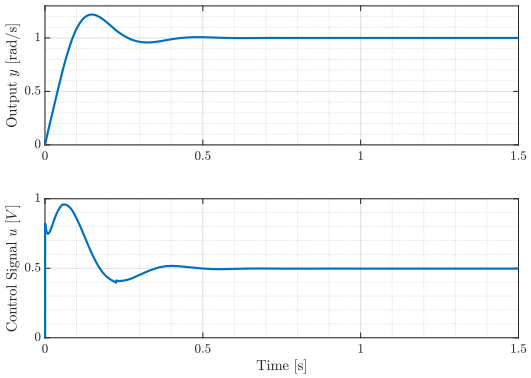
Figure 17: Plots showing location of poles and zeros for different tuning methods for the tilt velocity controller.

Performance Evaluation of Velocity Controllers

As with the position controller, the bandwidth of the closed-loop velocity controller systems are evaluated. The cutoff frequencies are found to be at approximately 11 rad/s for the pole placement based design, and approximately 28 rad/s for the Ziegler-Nichols based design. Therefore it can be concluded that the chosen sampling frequency of 1000 Hz is sufficient. The phase and gain margins are also examined for the velocity controller with the delay introduced to emulate the discrete controller. Using MATLAB, it is found that the gain for both design approaches is infinite, as the phase never crosses $-180°$. A phase margin of 145° for the pole placement based design and a phase margin of 50° for the Ziegler-Nichols based design is found. While the margin is significantly lower for the Ziegler-Nichols method, it is deemed sufficient.



(a) Pole placement based design.



(b) Ziegler-Nichols based design

Figure 18: Step responses of the tilt motor with velocity controllers designed using the pole placement method and the Ziegler-Nichols tuning method.

Looking at the step responses for the pole placement based and Ziegler-Nichols based controller

designs, shown in figure 18, it can be seen that as expected the pole placement based controller meets the requirement of no overshoot, while the Ziegler-Nichols method does not. It must be noted however, that the pole placement based design asks for a much larger control signal, u, which can lead to saturation issues when larger references are introduced.

7.3 Cascaded Controller Design

Figure 19 shows the principles of the cascaded controller design. Designing the inner controller, i.e. the velocity controller has been described in section 7.2. From the figure it can be seen that when designing the outer position controller, the inner controller has to be regarded as part of the plant.

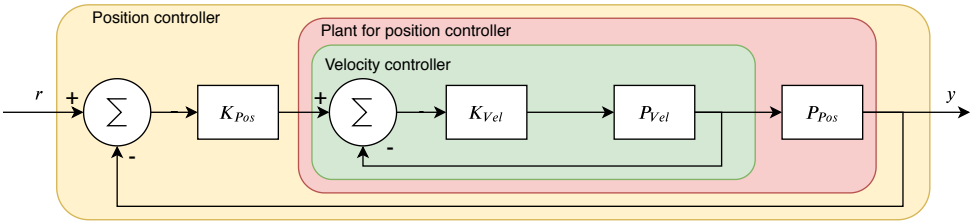
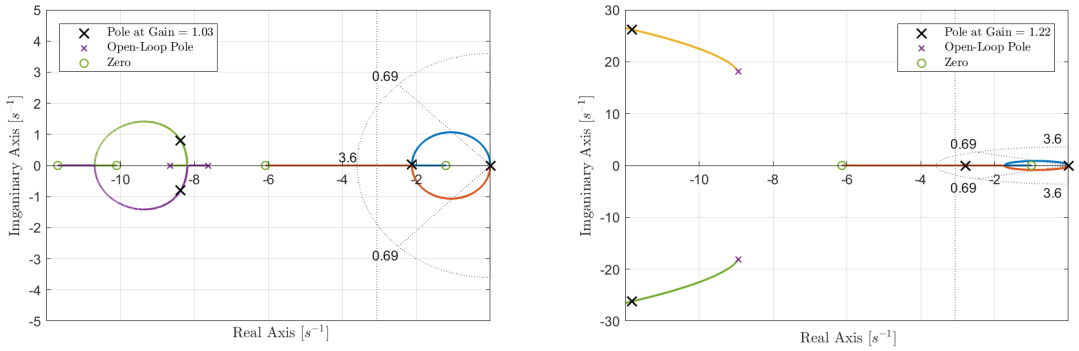


Figure 19: Block diagram of a cascaded controller. K_{Pos} being the position controller, K_{Vel} being the velocity controller, P_{Vel} returns the velocity of the plant and P_{Pos} returns the position of the plant.

Since it has been decided to focus on only designing velocity controllers for the tilt motor, it is only relevant to focus the cascaded design on the tilt motor. When considering the plant as the combination of the closed-loop velocity control system and an integrator to transfer from velocity to position, the design approach is identical as have been used throughout most of this section. Figure 20 shows that as expected, the poles and zeros stemming from the closed-loop velocity control systems, appear as open-loop poles and zeros in the root locus plot of the position control system. To design the PID position controllers, very similar zero placements methods are used. In the case of the pole placement based velocity controller, zeros are placed at $s = -6.1$ and $s = -1.2$ while for the Ziegler-Nichols based controller zeros are placed at $s = -6.2$ and $s = -1$. Even though the zeros stemming from the PID-controllers are placed almost identically, the root loci are quite different due to the different poles and zeros stemming from the velocity controllers. From the closed-loop pole locations, it is expected that the response from the control system based on the Ziegler-Nichols tuning will have a faster response, but also quite a bit more overshoot. From the bode plots to be found in the digital appendix, outlined in section 13 it can be seen that the sampling frequency of 200 Hz is sufficient. From the phase and gain margins plot, also to be found in the digital appendix, it can be seen that the Ziegler-Nichols based design has a slightly smaller phase margin at 75.7° to the 118° of the pole placement based design.

Looking at the step responses for the two designs, shown in figure 21, it can be seen that, ideally, both methods perform quite well in regards to the specifications in section 5, although the settling time requirement cannot quite be met. The Ziegler-Nichols based design gives slightly less overshoot, but does give an unwanted transient response. This is only the case when saturation is not

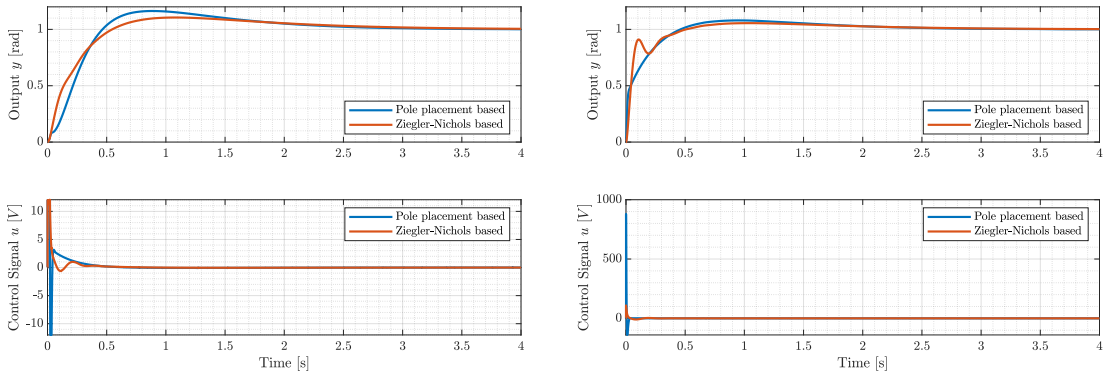


(a) Cascade control with velocity controller based on model.

(b) Cascade control with velocity controller based on Ziegler-Nichols.

Figure 20: Root locus plots for the cascade control systems based on velocity controllers designed using the model and Ziegler-Nichols.

considered. In the case when saturation is considered, the Ziegler-Nichols based design performs better in terms of overshoot, while both give a settling time which does not quite comply with the specifications.



(a) Systems with saturation included.

(b) Systems without saturation.

Figure 21: Bode plots of the closed-loop system with velocity controllers designed using the pole placement and the Ziegler-Nichols method.

7.4 Conclusion

Controller gains have been determined for PID-controllers based on different design methods. Based on a single controller design, position controllers have been designed for the pan and the tilt motors. This has been done purely based on the mathematical models of the motors. Furthermore, velocity controllers have been designed for the tilt motor, allowing for the design of a cascaded design. One velocity controller has been designed based on the model and another based on the heuristic Ziegler-Nichols method. Using the designed velocity controllers as inner loops, cascaded position controllers have been designed.

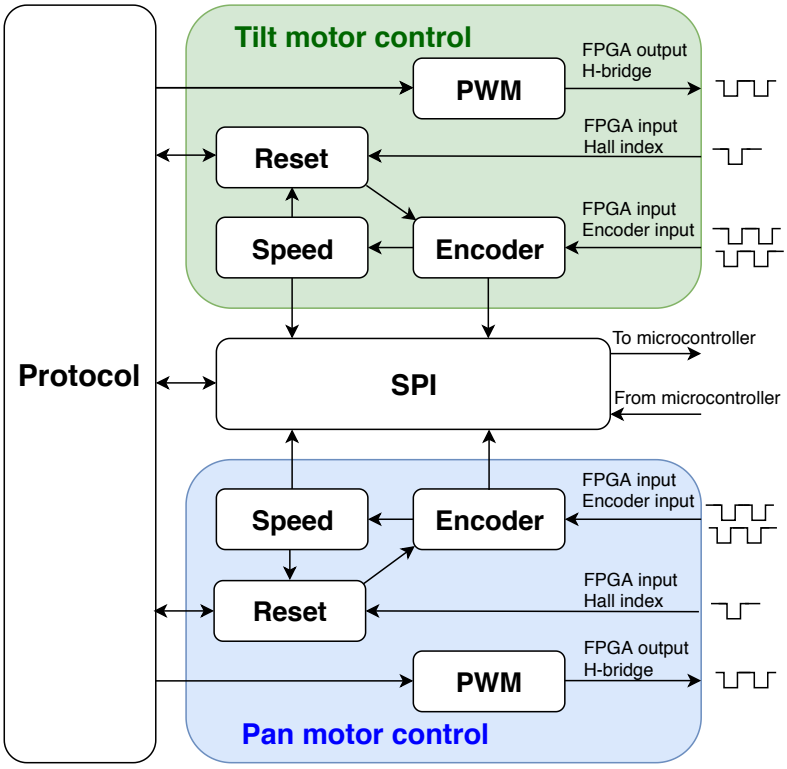


Figure 22: Overview of the relation between the implemented modules on the FPGA. Arrows indicate direction of information flow between modules.

8 System Integration and Implementation

This section will concern the design and implementation of the functionality on the FPGA as well as the microcontroller. The FPGA is to handle the PWM signal, direction of the motors and the data from the encoders used for the position and velocity feedback. The microcontroller is to handle the computation of the PID-controllers, SPI communication and user input..

8.1 FPGA

The logic implemented in the FPGA is divided into modules, which will be described in this section. Figure 22 illustrates the modules implemented on the FPGA and the relation between the different modules. Speed, Encoder, PWM and Reset modules are implemented twice, one for each motor. Flowcharts describing the logic in the different modules are provided in the digital appendix, outlined in section 13.

Encoder Module

Each motor is equipped with a motor encoder, which is interfaced with an encoder module on the FGPA. The Encoder module takes pulses as input from the motor encoders and outputs a signed value based on the number of encoder counts indicating the motor position.

The Encoder module looks at channel A and B from the motor encoder continuously, and updates the position whenever a falling or rising edge occur on either of the channels. The position is either incremented or decremented depending on the turn direction of the motor, indicated by the pattern of encoder edges. As the pulses from the motor encoder only can be used to get relative positions, the Encoder module needs a signal to reset the output value to zero. This signal is provided by the Reset module.

Speed Module

The Speed module takes the position provided by the Encoder module as input. The module outputs a signed value representing the angular velocity of the motor.

There are two ways to determine the angular velocity from encoder readings. It can either be determined by evaluating encoder counts within a constant time period, providing more precise readings when the angular velocity is high. Another approach is to count time periods between encoder counts, providing more precise readings when the angular velocity is low. In this project the latter method is chosen. This is implemented by providing the Speed module with a clock of 100 kHz. The angular velocity is calculated based on the assumption that change in position of the motor output shaft per encoder count is constant. Later this assumption was discovered to be wrong, the consequences will be discussed in section 10.

For the system to know if the angular velocity of a motor is zero, a zero-flag is implemented. If no change in encoder position is detected within a time period of 300 ms, the zero-flag is raised.

PWM Module

The PWM module takes a 10-bit duty cycle reference and a direction signal as input. As output it provides signals that interface with the H-bridges on the pan-tilt system. These signals comes in the form of a 10-bit phase-corrected PWM-signal at a frequency of approximately 48.8 kHz along with a signal for inverting the polarity in the H-bridge to control the direction. Figure 23 shows a flowchart describing the process in the PWM module that generates the PWM-signal.

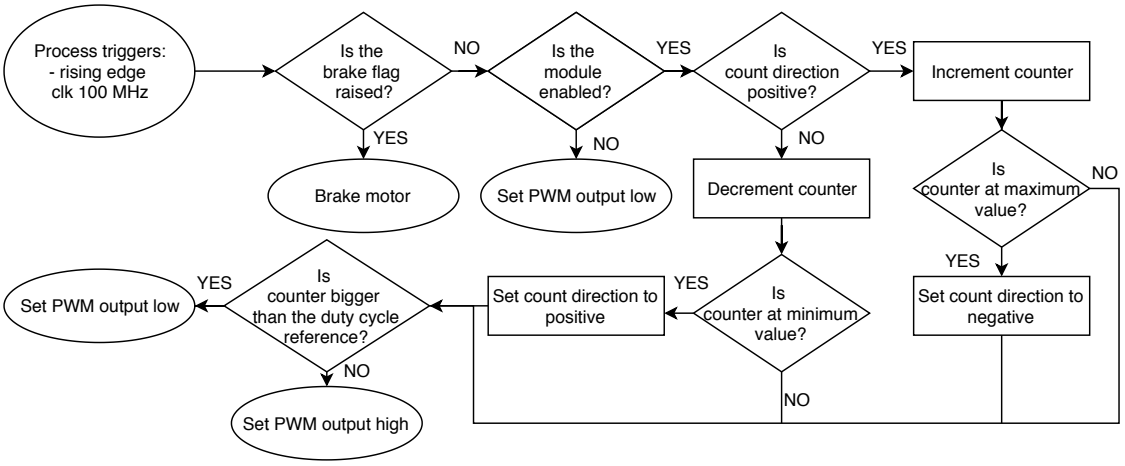


Figure 23: Flowchart describing the process in the PWM module which outputs a PWM-signal according to a duty cycle reference.

Reset Module

The purpose of the Reset module is to provide the Encoder modules with a reference position. The Reset module takes an index signal and the zero-flag provided by the speed module as input. The output of the Reset module is a signal used to reset the Encoder modules associated with the pan motor and tilt motor.

When a reset routine is enabled from the Protocol module, the motors drives the pan and tilt frames with constant low duty cycle, until a pulse from the respective index hall sensors are received. If the pan frame is stopped during a reset, due to physical limitations in the setup, the module reverts the direction of rotation. The physical limitations are detected when the zero-flag is signaled from the Speed module. When it receives the pulse from the index hall sensor, the Reset module brakes the motors while pulsing the Encoder module to reset its position counter to 0.

Protocol Module

The Protocol module acts as a control unit activating different modules in the FPGA depending on commands received from the microcontroller. The Protocol module takes a duty cycle reference from either the reset module or SPI module as input. The Protocol module provides the PWM module with the correct duty cycle reference, determined by the commands from the microcontroller.

Initially, the module was designed to be able to provide the microcontroller with states, thus functionality to inform the microcontroller of the states of the pan-tilt system is provided. In addition to that the protocol module enables the possibility of expanding the functionality of the FPGA for future use.

SPI Module

The SPI module on the FPGA is responsible of interfacing with the microcontroller. On figure 24 an overview of the SPI module is provided. The module contains seven internal SPI slaves. Four slaves are responsible for transmitting the feedback information used for the PID-controllers in the microcontroller. Two slaves are used for receiving the duty cycle references sent from the microcontroller. The last slave is used for the Protocol module.

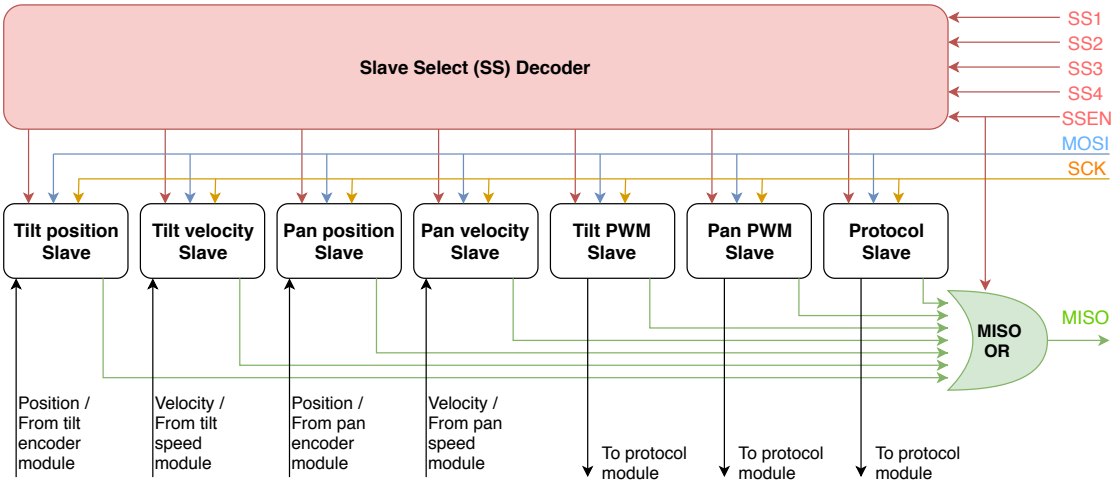


Figure 24: Overview of the SPI module implemented on the FPGA, showing the seven slaves, the slave-select-module and logical MISO OR-module used to OR the output of all the slaves. All named signals on the right side of the figure interface with the microcontroller, which is the SPI Master.

The microcontroller is the master device controlling the slaves. To enable the slaves, as seen on figure 24, SSEN is used. To access a specific slave a 4-bit address is used, where the signals SS1-SS4 specifies the address allowing the Slave Select Decoder to select the slave. Master-Out-Slave-In (MOSI) transfers data from the microcontroller to the slaves. The MISO OR gate is implemented due to internal tri-state buffers not being available on the FPGA. The Master-In-Slave-Out (MISO) signal line transfer data to the microcontroller. A data transfer is controlled by the Serial Clock (SCK) by shifting data from the slave to the master and vice versa at the same time. The SCK-signal is controlled by the microcontroller, thus the FPGA only has to keep position and velocity slaves updated with the newest position and velocity feedback data.

8.2 Microcontroller

It is imperative that the microcontroller is in control of the overall timing of the system, in order to guarantee a consistent timing of the controllers. Therefore, the microcontroller is responsible for pulling data from specific slaves on the FPGA using SPI. As described in section 5, it is decided to accommodate a cascaded controller design consisting of two PID-controllers on each motor. This yields a total of four tasks needed to be scheduled regarding the motor control. Furthermore

the microcontroller needs to handle the user interaction. The program will be structured as seen on figure 25. The task diagram is simplified displaying only one of the cascaded controller for controlling only one motor, as the second cascaded controller is identical in its structure. A more detailed task diagram can be found in the digital appendix section 13. In FreeRTOS tasks have assigned a priority, meaning that high priority tasks are ensured to be scheduled and completed before tasks of lower priority. To achieve a real-time system prioritisation of tasks is vital.

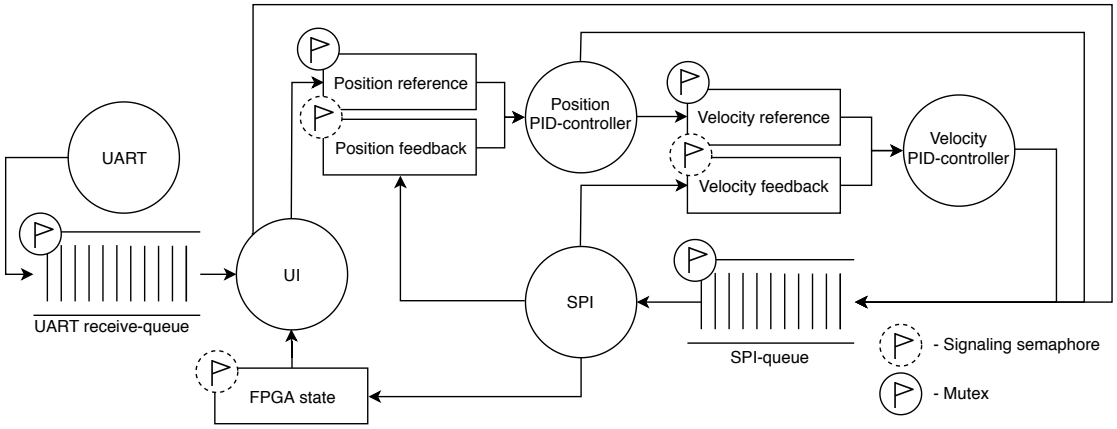


Figure 25: Simplified task diagram showing the different tasks running on the microcontroller. Circles represent tasks and squares represents buffers.

PID-controller Tasks

The PID-controller tasks on the microcontroller are responsible for calculating the control signals, which are used to control the duty cycle references. As seen on figure 25 each controller needs three state buffers, two for input and one for the output. The velocity controller outputs to the SPI-queue. The PID-controller task is responsible for requesting feedback data when it is needed. To request feedback data, the controller task needs to signal the SPI task via the SPI-queue. When the feedback data is available, the controller task is signaled with a semaphore connected to the feedback state buffer. When the controller is waiting for feedback data to be updated, the task enters the blocked state, where it uses no CPU resources.

The overall flow of the PID-controller task can be seen on figure 26. All controller tasks implemented follow the same general structure, however, the coefficients used to calculated the PID-terms differ. When the feedback data is received, the error between the reference and feedback value is calculated. Because the derivative term of the PID-controller amplifies noise, it is beneficial to include a low pass filter in the implementation [5]. The filter can be used specifically on the derivative term, or be applied to the error signal passed to all three terms. Initially the latter approach was implemented, but through tests, it was found that an extra filter specifically on the derivative term was needed as well. Designing these filters has not been highly prioritised and thus not much thought has gone into the filter design. This has been discovered to possibly be a poor prioritisation, and will be discussed in section 10. The filters are designed using the MATLAB

Filter Designer. The filter design seen in equation 22,

$$H(z) = \frac{1}{0.0249z^2 + 0.9502z + 0.0249} \quad , \tag{22}$$

is used for the general error and the design shown in equation 23,

$$H(z) = \frac{1}{0.0555z^5 + 0.1666z^4 + 0.2777z^3 + 0.2777z^2 + 0.1666z + 0.0555} \quad , \tag{23}$$

is used for filtering the derivative term. A Infinite Impulse Response (IIR) filter has also been considered. However since a IIR filter can become unstable, no further time was spent on this idea. In hindsight this might not have been the best choice, why will be discussed in section 10.

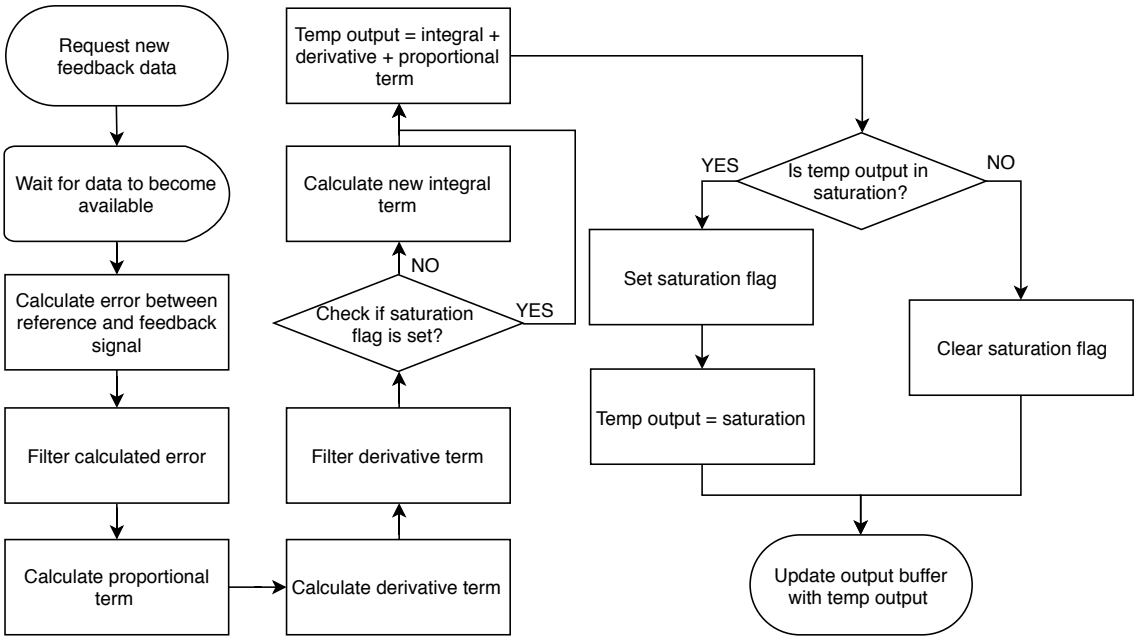


Figure 26: Flowchart describing the code in the PID-controller tasks on the microcontroller.

To make sure the integral term does not wind up when in saturation for an extended period of time, anti-windup by conditional integration is implemented. As seen on figure 26 the controller performs a saturation check before updating the output buffer. If the calculated output is in saturation, a flag is raised and the output is set to either the negative or positive saturation value, depending on the sign of the error. On the position controller the saturation limit is equal to the maximum angular velocity, found by test to be approximately 22 rad/s, of the pan-tilt system. The saturation limit of the velocity controller is equal to the maximum voltage possible to supply the motors which is ± 12 V.

SPI Task

For the PID-controllers to be able to get their respective feedback data, an SPI task is necessary. The task is responsible for processing requests coming from each controller, receive data from the seven different slaves and put it in the correct state buffer. To do this each controller task and the

belonging feedback state buffer is associated with a slave ID. The ID is connected to a slave on the FPGA, containing data only relevant to the requesting controller. The tilt PWM slave and pan PWM slave do not have associated state buffers, since the data received via these connections is not used.

The microcontroller has built-in hardware that enables transmitting and receiving data using SPI. To enable the slaves, a hardware controlled slave-select pin connected to SPI is used. To select a specific slave, four general purpose input output (GPIO) pins on the microcontroller are used. By setting the pins high or low, each slave can be selected by their 4-bit address as mentioned in section 8.1. Data is transferred at a bit rate of 13.3 Mbit.

As default the SPI task is in the blocked state. Only when a request enters the SPI-queue the SPI task is given CPU time. As seen on figure 27, the task deciphers the request and enables the correct slave after which data is transmitted and received. If the slave ID has an associated state buffer, the data received from the transmission is put in that state buffer, while signalling the semaphore for the PID-controller sending the request.

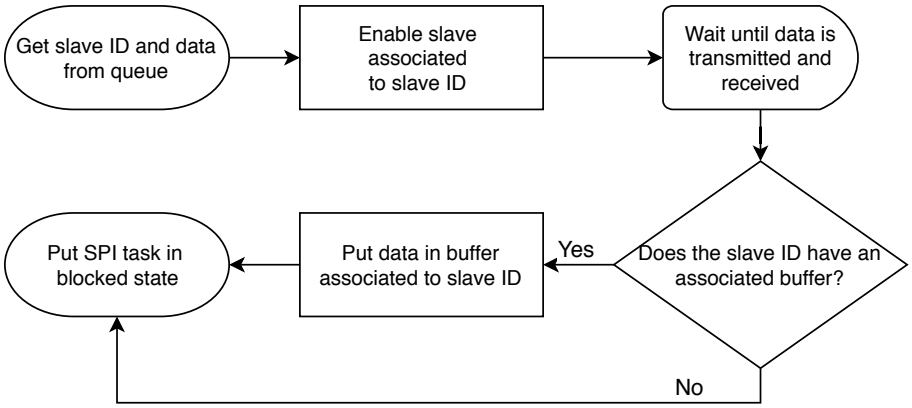


Figure 27: Flowchart describing the code execution in the SPI task on the microcontroller.

As the PID-controllers are designed based on standard SI-units, it is necessary to perform a conversion to make sure the feedback complies with this standard. The relation between the angle of the motor measured in radians, and the corresponding number of encoder counts is $\pi/180$. As a result the conversion is given by table 3.

Conversion to an angle [rad]	Conversion to an angular velocity [rad/s]
$\theta_m = \frac{\pi}{180} \cdot \theta_{\text{FPGA}}$	$\dot{\theta}_m = \frac{\frac{\pi}{180}}{\dot{\theta}_{\text{FPGA}}} \cdot 10^5$

Table 3: Unit conversions for data received from position and velocity slaves.

The angle of the motor, θ_m, is calculated using the the encoder count value, θ_{pos}. The angular velocity feedback, $\dot{\theta}_m$, is calculated based on $\dot{\theta}_{\text{FPGA}}$, provided by the Speed module on the FPGA.

UI Task and UART Task

To enable a user to control the pan and tilt frames to desired positions, a user interface (UI) and UART task is necessary. The UART task is responsible of receive data from a PC terminal through UART and put it in a queue from where the UI task consumes the data. From here a protocol implemented to interpret the received data and control the pan-tilt system accordingly. An FPGA state buffer was initially thought of to save information about different states of the FPGA. From here the UI task could pass information on to the user, but it was deemed a low priority and no further time was spent on this.

The UART task is assigned the lowest priority, meaning it is given CPU time when no other tasks are scheduled. When it is running, it checks if user input has been received. If so, data is put into the UART receive-queue, signalling the UI task. The UI task checks which command the received data corresponds with and acts as described in table 4. To make it more intuitive when the user uses the, *Go to position* command, the desired absolute positions of the frames are entered in degrees instead of radians. The conversion is handled by the UI task as seen on equation 24,

$$pos_{radians} = \frac{pos_{degrees} \cdot 3}{180} \cdot \pi \tag{24}$$

where $pos_{radians}$ is the reference point, that goes to the pan-tilt position controllers.

Command	Number of bytes	Description
Reset	1	Begin the reset command.
Home motor	2	The tilt and pan frame returns to index position.
Go to position	6	Makes the tilt or pan frame go to a specific position.
Abort	1	Shuts down controller task.

Table 4: UART Protocol showing the different commands available for the user of the system. Number of bytes indicate how many characters is needed to pass the command, a detailed guide for user operation is included in the digital appendix outlined in section 13.

8.3 Conclusion

Logic has been implemented on the FPGA to evaluate the position and speed of the motors, to provide a PWM-signal to control the motors and to communicate over SPI with the microcontroller. On the microcontroller, FreeRTOS a real-time operating system, has been used for implementing four PID-controller tasks. For the controllers to get their respective feedback data, an SPI task communicating with seven slaves has been implemented. The embedded system also includes an UART task to pass on user commands to an UI task, which on the basis of a developed protocol, controls the pan-tilt system.

9 Tests

This section will introduce the tests prompted by the requirements in section 5 and present the results. These results will determine whether it has been possible to comply with the requirement specifications. Additional raw data from the tests can be seen in the digital appendix.

To ease access to data from the FPGA during tests, a *MicroBlaze* has been implemented, which utilises UART to communicate directly with a computer. This allows for direct access to the position and velocity of the pan and tilt motor making data processing easier.

9.1 Microcontroller Timing Test

Running multiple tasks on one microcontroller it is essential to know, how much of the CPU time is utilised. If the CPU is overloaded the risk of starvation is introduced. In addition it is wanted to examine the processing time of the controllers on the microcontroller, to be able to deduce the overhead, thus it is wanted to test whether the system can satisfy the requirement from section 5:

- The PID-controller tasks must not utilise more than 60 % of the CPU time on the microcontroller.
- It must be verified that the overhead does not compromise the timing of the system.
- A maximum execution time for the PID-controller must be determined.

To estimate a maximum allowed processing time for a controller, the maximum time for which the PID-controller tasks may utilise the CPU time is divided into four parts. With a maximum CPU time utilisation of 60 %, and the possibility of four concurrent controller tasks executing at a rate of 1000 Hz, a single controller is to be processed within a time period of 150 µs.

The single controller test is performed with the velocity controller described in section 8 using a fifth order filter on the derivative term. The CPU utilisation test is executed with four controllers, the UART task and the SPI task scheduled by FreeRTOS. The UI task is not taken into account, because no user input is required for the test. The position controllers are scheduled at a frequency of 200 Hz and the velocity controllers at 1000 Hz. The timing of the tasks are done by driving GPIO pins high upon start and low upon end of the task cycle of all individual tasks, thus enabling the possibility of timing the tasks with an oscilloscope.

Results

Table 5 shows the timing of running the controllers including the UART, SPI and UI task. T_c is the duration between the first task starts to the same task starts again, as seen on figure 28. t_1 is the duration of two velocity PID-controller tasks and t_2 is the duration of two position PID-controllers tasks. The UART task is not directly included in the calculations, however, as seen on table 6, it

is so short, that it is deemed negligible. The SPI task is called directly from the PID-controller tasks, thus indirectly included in the calculations. This makes it possible to estimate the average CPU utilisation, $U_{CPU,avg}$, of the PID-controller tasks as seen in equation 25,

$$U_{CPU,avg} = \left(\frac{t_{1,avg} + t_{2,avg}}{T_{c,avg}} \right) \cdot 100 = \left(\frac{457.6\,\mu\text{s}}{992\,\mu\text{s}} \right) \cdot 100 \approx 46.13\,\% \quad , \tag{25}$$

where $t_{1,avg}$ and $t_{2,avg}$ are the average duration of t_1 and t_2 and $T_{c,avg}$ is the average time of T_c as seen on figure 28.

	Test number				
	1	2	3	4	5
$t_1 + t_2$ [µs]	458	456	456	460	458
T_c [µs]	990	988	994	994	994
U_{CPU} [%]	46,3	46,1	45,9	46,3	46,1

Table 5: Data regarding CPU utilisation of the four PID-controller tasks and the SPI task.

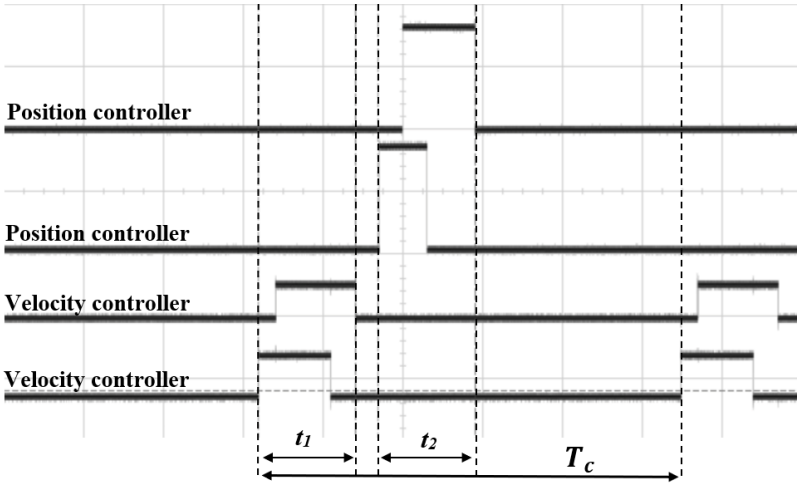


Figure 28: Illustration of the scheduling of the four PID-controller tasks, the UART and SPI tasks. t_2 is the duration of both position PID-controllers run at a rate of 200 Hz. t_1 is the duration of both velocity PID-controllers run at a rate of 1000 Hz. T_c is the duration between the first task starts to the same task starts again.

Table 6 shows the timing of the UART and the velocity PID-controller task respectively. Deduced from the data presented in the table, it can be calculated that the PID-controller can be processed in an average time of 107.8 µs with an fifth order filter on the derivative term. In accordance with the requirement specification this is sufficient for a single controller.

	Test number				
	1	2	3	4	5
UART [ns]	458	500	500	500	501
Velocity Controller [µs]	107,8	107,8	108	107,8	107,8

Table 6: Duration of the UART task and velocity PID-controller task.

It is possible to get an estimation of the overhead of the system by comparing the values of table 6 and 5. This is done by comparing the total time of executing four PID-controllers with the time of processing a single PID-controller. This yields an estimated overhead in terms of CPU time as seen in equation 26.

$$\text{overhead} = \frac{T_c - (T_{controller} \cdot 4)}{T_{c1}} \cdot 100 \approx 5.7\,\% \qquad , \tag{26}$$

where $T_{controller}$ is the processing time of a single velocity controller.

9.2 SPI Stress Test

Using SPI as the only communication between the FPGA and the microcontroller, it is essential to test the reliability of the system to ensure that no information is lost during transmission. It is wanted to test the requirement stated in section 5:

- Using SPI with a bit rate of 13.3 Mbit, the FPGA is to receive 99.9 % messages correct with no errors.

The test was preformed by sending 10000 messages from the microcontroller to the FPGA. Each message contains the value of a counter, starting at 0 and incrementing to 9999. Each message is subsequently transmitted to a computer with the help of the MicroBlaze, and processed to check if all values are present and in the correct sequence.

Results

As seen in table 7 the five test resulted in the same outcome, with all messages being received and in the correct sequence.

Test no.	Bit rate [Mbit]	Messages sent	Messages received
1	13.3	10000	10000
2	13.3	10000	10000
3	13.3	10000	10000
4	13.3	10000	10000
5	13.3	10000	10000

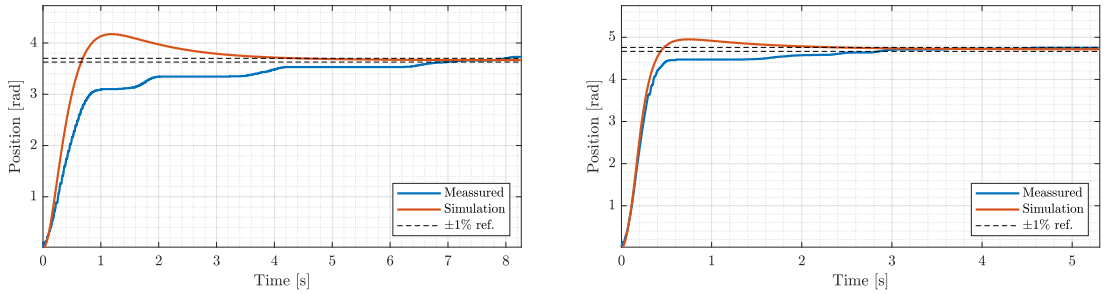
Table 7: Data regarding the SPI stress test.

9.3 Test of Controller Designs

This section concerns the test of the PID-controllers using different methods for tuning and designs. The parameters used in each test is shown in table 2, and are designed using the design guides of 5 % overshoot, 1.5 s settling time and 0.5 s rise time.

Single Position Controller using Pole Placement

The tests are executed with a reference point of $\theta = \frac{7\pi}{6}$rad for the pan motor and $\theta = \frac{3\pi}{2}$rad for the tilt motor. As seen on figure 29a the measured response has a slower step response compared to the simulated response. Furthermore there are stationary points in the measured response, showing that the pan frame has stopped moving. On figure 29b the measured response resembles the simulated quite well. The data for the tests can be seen on table 8. Applicable for both measurements is that they have close to no overshoot and longer settling times than the simulated and specified in the requirements.



(a) Pan motor.

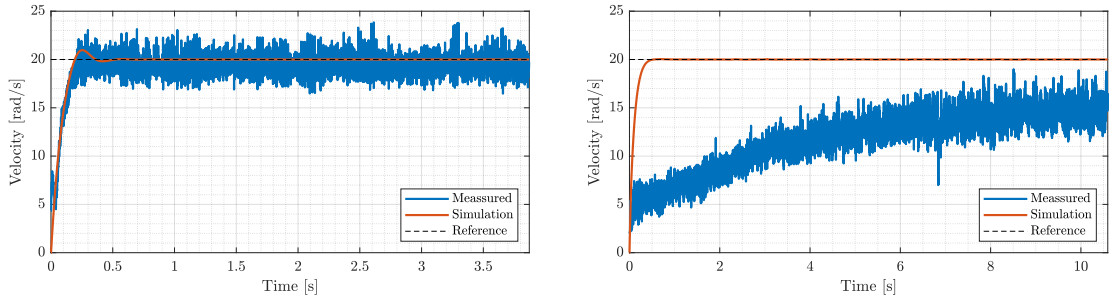
(b) Tilt motor.

Figure 29: Step response of the position using a single controller with pole placement. The measured response is an average of five tests.

Velocity Controller for the Tilt Motor

The velocity controller is tested using gains found from the Ziegler-Nichols and pole placement methods on the tilt motor. A reference point of $\dot{\theta} = 20\,\text{rad/s}$ is used. As seen on figure 30a the

measured response initially resembles the simulated signal using the Ziegler-Nichols method. Seen on figure 30b, using pole placement a very slow response compared to the simulated is achieved in addition to a steady state error. Studying data from figure 30a further it appears like a small steady state error is present as well. The reason for this behaviour is discussed in section 10. Applicable for both tests is that the measured signals are corrupted by noise. Data regarding the performance of the controllers are presented in table 8.



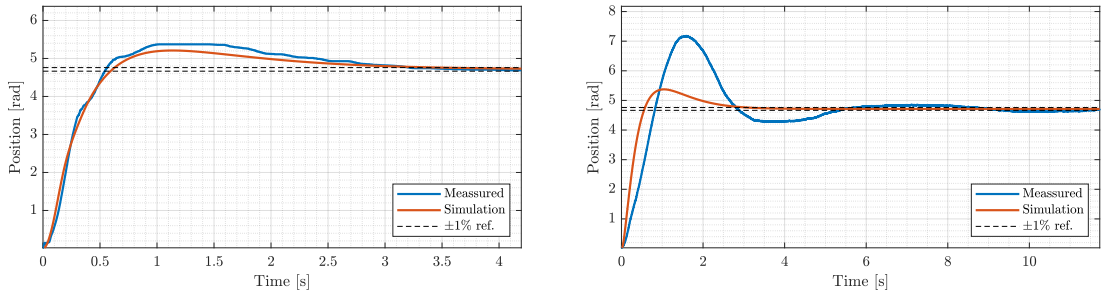
(a) Ziegler-Nichols method.

(b) Pole Placement method.

Figure 30: Step response of a single velocity controller on the tilt motor. The measured response is an average of five tests.

Cascaded Position Controller for the Tilt Motor

The cascaded PID-controller design is tested using the velocity controller gains analysed previously and the position controller gains found using the pole placement method with the velocity controllers taken into account. A reference point of $\theta = \frac{3\pi}{2}$rad is used. Figure 31a shows that using the Ziegler-Nichols method, the measured signal resembles the simulated response quite well. However using the pole placement method, illustrated on figure 31b, the measured response has a lot more overshoot than the simulated response. It could appear that this is due to the poor velocity controller response, which is discussed in section 10. The performance results are presented in table 8.



(a) Ziegler-Nichols method.

(b) Pole Placement method.

Figure 31: Step response of a cascaded PID-controller for the position of the tilt motor. The measured response being an average of five tests.

Cascaded Position Controller for the Pan Motor

The cascaded controller on the pan motor is executed using a reference point of $\theta = \frac{7\pi}{6}$. Since no cascaded controller has been designed for the pan motor, the controller gains are identical with the ones used for the cascaded tilt controller based on the Ziegler-Nichols method. This is done mainly as means to evaluate the accuracy of the pan motor model, by comparing the measured response with the simulated response. This is also done to assess the importance of customising the gains for the specific motor. According to figure 32, the measured signal seems to have a slower response than the simulated response. However neither the simulated nor the measured response are considered ideal, which highlights the importance of tailoring the gains for the specific motor.

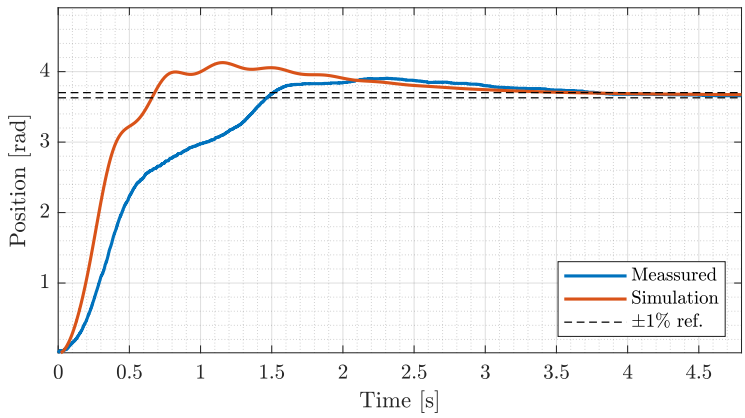


Figure 32: Position step response using a cascaded PID-controller for the pan motor. The measured response being an average of five tests.

Results

Table 8 shows the data regarding the performance of the different PID-controllers. As it is seen none of the designed controllers satisfy the specifications of less than 5 % overshoot, 1.5 s settling time and 0.5 s rise time, however some controllers achieved better results than others. The single position PID-controller seems to have the overall best performance in relation the the requirement specifications. The reason for the deviations comparing the simulated and the measured responses are discussed in section 10.

Controller	Motor	Method	Overshoot (%)	Settling Time (s)	Rise Time (s)
Position	Pan	Pole placement	14.3	8.27	1.74
Position	Tilt	Pole placement	0.8	2.86	0.34
Velocity	Tilt	Ziegler-Nichols	19.2	N/A	0.15
Velocity	Tilt	Pole placement	N/A	N/A	7.8
Cascaded	Tilt	Pole placement	52	11.47	0.61
Cascaded	Tilt	Combined	14	3.22	0.38
Cascaded	Pan	Combined	6.6	3.78	1.13

Table 8: Step response specifications of the different controllers tested. Entries marked with N/A could not be determined from the available data. The controllers using the combined method is tuned using Ziegler-Nichols for the inner loop and pole placement for the outer loop.

9.4 Conclusion

It has been possible to achieve an average CPU utilisation time for the PID-controllers of approximately 46.13 %, thus satisfying the project stated requirement of the PID-controllers utilising a maximum of 60 % of the CPU time. A maximum allowed processing time for a single PID-controller is determined to be 150 µs. Furthermore the processing time of a single PID-controller is 107.8 µs, thus satisfying the requirement. An average overhead is estimated to approximately 5.7 %, hence not deemed to compromise the timing of the system.

The SPI communication has been tested to satisfy the requirement of transmitting at a bit rate of 13.3 Mbit with a success rate of 10000/10000 messages successfully received, thus complying with the requirement of a success rate of 99.9 %.

The Ziegler-Nichols and pole placement design methods have been tested. The methods are tested on single PID-controllers as well as PID-controllers in cascade.

10 Discussion

During the project, choices have been made with regards to design of controllers and implementation on the given devices. Some of these choices, both due to inexperience in the disciplines and due to initial project conditions, have proved to be inadequate towards meeting the system response requirements decided by the project group in section 5. In this section the results as well as the choices and alternatives will be discussed.

10.1 Evaluating Performance

Looking at table 8 it is seen that the specifications established in section 5 have not been completely met by any of the implemented controllers. In this regard, it must be acknowledged that none of

the simulations meet all the specifications either. A more experienced control engineer might have settled on less strict requirements. Over all it has been possible to achieve some performances which are not too dissimilar from those that were found through simulation. It is likely that manually tuning the parameters of the controllers, based on the values presented in this report, could improve the performance. Had more time been available, this would have been interesting to study further. After all, one of the main advantages of the PID-controller is the fact that it is simple to tune. It would also have been interesting to explore the effects of using the modern state feedback method introduced in section 3, and compare the performance results to those achieved using the PID-controllers.

It was expected that the cascaded controller design would yield better performance when implemented compared to a single controller, since it theoretically offers better rejection towards noise. However, with the chosen designs the cascaded controller actually performed worse than the single controller designs in both simulations and implementation. This is likely because the velocity controllers should have been designed to have a faster response. For cascade control to make sense, the dynamics of the inner loop should be at least five times faster than the dynamics of the outer loop. Furthermore, the velocity controller should have had a high gain to limit disturbances. Had these requirement been properly met in designing the velocity controllers, the cascade control performance may have been better. It should also be mentioned that it has later been discovered that using a full PID in an inner loop is unconventional. Usually either PI or just P control is used for inner loops [14]. It was found that the cascaded controller which relied purely on the mathematical model of the system performed drastically worse than the one which made use of the Ziegler-Nichols tuning method. This is not unexpected, since it is acknowledged that several imperfections in the mathematical model of the pan-tilt system exist. Therefore a heuristic approach such as the Ziegler-Nichols is likely to yield equally good or better results compared to the pole placement based approach. Considering figures 29b, 30a and 31a it can be seen that, the response from the physical system is quite close to the simulated response of the tilt motor. This indicates that the mathematical model of the tilt motor is somewhat accurate. The big deviations which are seen in figure 30b and 31b are likely caused by another issue which is discussed later in section 10.3. Figure 29a and 32 shows that a lot more deviation is found for the model of the pan motor. This makes sense since the parameters, which are found experimentally in section 6.1, are based solely on the tilt motor. If disassembling the pan-tilt system was allowed, it would be interesting to perform similar tests to determine the parameters of the pan motor. From looking at the two figures it also appears that the pan motor generally has a slower response than what the model predicts, which among other things could indicate that the estimation of the inertia of the pan motor is too low. This is likely to be true since the simplifications, introduced in section 6.1, will yield a slightly lower result for the inertia especially for the pan motor. To more thoroughly analyse and present the differences between the simulations and the physical setup, it would have been beneficial to extract and plot the control signals produced by the implemented PID-controllers.

10.2 Encoder Accuracy and Resolution

As the angular velocity is deduced from the encoder output, the performance of the velocity PID-controllers and the accuracy of the encoders is closely linked. As seen in figure 30 the measured velocity response is very noisy. The accuracy of the motor encoder has been recognised as one of the possible sources of this noise. The accuracy of the motor encoder is inadequate, as the time between encoder edges at a constant angular velocity is not constant. Table 9 shows the time period between consecutive encoder edges. The data is sampled from the tilt motor encoder, over a period of 14 encoder edges, while a constant voltage is applied to the motor. It is acknowledged that the angular velocity is uneven through one revolution of the tilt frame. It is presumed that this velocity variance can be neglected due to the short time frame of the 14 edges compared to the 1080 for a full revolution. The percentage increase from the minimum time period to the maximum time period is 59.6 % when looking at the data in table 9. The difference is significant. In this section the consequences of this encoder behaviour, the fact that little is done to compensate for the behaviour and what could have been done is discussed.

Time period (ms)												
1.28	1.52	1.42	1.4	1.34	1.66	1.04	1.66	1.26	1.38	1.34	1.6	1.22

Table 9: The time period between encoder edges measured in milliseconds.

One way to compensate for the inconsistency would be to group four edges of the encoder signals into one count. This would result in percentually less variance between counts than what is experienced with the current implementation. With this modification the resolution of the encoder module would be 90 counts per revolution, four times lower than the original resolution, which is not desired. This would introduce other problems for the PID-controllers implemented, as the rate at which new data is available would decrease.

As mentioned in section 8.1, another way of determining the angular velocity would be to count encoder edges in a constant time period. Possibly this method compensates for the inaccuracy of the motor encoders better than the method utilised, as the variance in the time between encoder edges becomes less significant when the amount of encoder edges between a time period is high.

Currently, the PID-controllers requires the angular velocity value from the Speed module every 1 ms. This requires the Speed module on the FPGA to update the speed value with a frequency of at least 1 kHz. With a resolution of the motor encoder at 1080 encoder counts for one revolution and one revolution taking approximately 5 seconds for the tilt frame at its slowest speed, this yields an average of 0.2 encoder counts per sample. This would not be adequate for calculating the angular velocity.

The optimal solution would probably be to replace the current motor encoders with encoders with much higher resolution and better precision. Encoders with resolution in the 1000- and 10000-range are easily available. An encoder with better resolution would make it more viable to count encoder edges within a constant time period, even at the lowest possible speed.

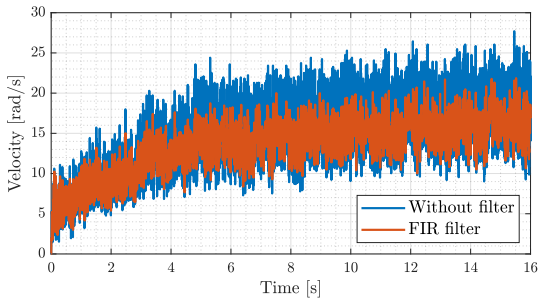
In this regard, it is relevant to address the rate at which data is sampled, and thus the rate at which the PID-controllers should execute. From the beginning of the project it was determined to execute each controller at a rate of 1000 Hz for single and inner controllers, and 200 Hz for outer controllers. From the bandwidth of the closed-loop controller systems presented in section 7, it is indicated that execution rates as low as 100 Hz may be viable. The fast rate was chosen because it was assumed that a faster sampling rate would give a more accurate feedback signal. As it has been analysed in this section, the sensor hardware significantly limits the rate at which sampling makes sense. Furthermore, as PID-controller execution rate increases, so does the noise sensitivity of the derivative term. Based on this, it may have been of interest to experiment with slower execution rates.

10.3 The Filtering

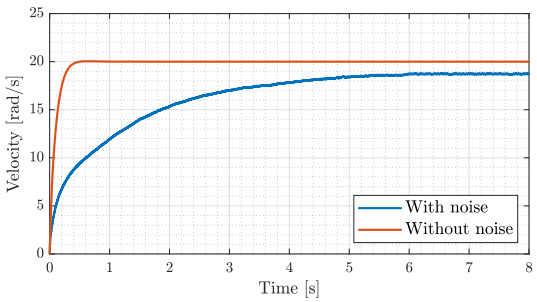
As described in section 8.2, filtering the feedback data was at first not assumed important to achieve good system response. With the very noisy velocity feedback data presented in figure 30a and 30b however, it is apparent that filtering is particularly important in this system. In neither of the figures does the system converge to the reference even though integral terms are implemented to correct this. Simulations, presented in figure 33b and 34b, indicate that the deviation from the reference could be due to noise affecting the derivative term.

It is wanted to evaluate the effect of the implemented fifth order FIR filter. In figure 33a, the implemented filter, described in section 8.2, is applied to the raw velocity feedback data. The figure shows a filtered but noisy signal and from the tests in section 9, it is obvious that the controller does not reach the reference.

Trying to understand how noise affects the response of the system, a step response is simulated where a high frequency sine wave is imposed on the feedback signal to emulate the noise. The simulation is shown in figure 33b and is based on the setup used for the pole placement based velocity controller test described in section 9.3.



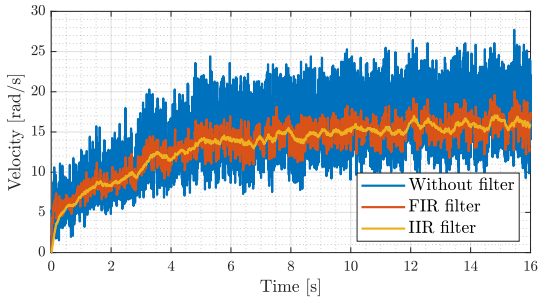
(a) Fifth order FIR filter applied to raw velocity feedback data.



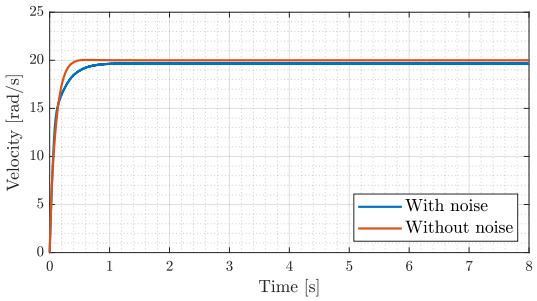
(b) Simulation of the system with an added sine wave as noise.

Figure 33: Step responses that shows the effect of a fifth order FIR filter and the effect that noise has on a response.

From figure 33b it is clear that the noise increases the rise time and increases the steady state error. On figure 34b a filter is implemented on the derivative term with transfer function $H(s) = \frac{5}{5+s}$. With the noise components at higher frequencies being dampened, the derivative term performs much better allowing for faster response and the system almost converging towards the reference at 20 rad/s. This emphasises that it would have been favourable to put more time into deliberately designing a filter, in order to accomplish a response with less error when noise is present.



(a) 20^{th} order FIR filter and a first order IIR filter applied to raw velocity feedback data.



(b) Simulation of the system with added noise and filter on the derivative term.

Figure 34: Step responses that shows the effect of more powerful filters and the effect that a filter applied to noise has on a response.

On figure 34a a 20^{th} order FIR filter is applied to the raw velocity feedback data. Even though the amplitude of the data is even smaller than with a fifth order filter, it is still noisy. A test would be necessary to see if the response would meet the chosen requirements. The figure also shows that using an IIR filter, substantially better noise reduction can be achieved using only a first order filter. Implementing an IIR filter may have been beneficial to achieve better performance.

10.4 Using Built-in Hardware in the Microcontroller

The project description states that the FPGA must be used for interfacing with the pan-tilt system, while the microcontroller must handle the user input and that these two devices must communicate to each other through SPI. However the processing power available on the microcontroller alone or the amount of logic blocks available on the FPGA alone far exceeds the requirements by the scope of the project. Therefore it is possible to implement the project on just one of the devices. It is clear that the project is constructed in this way to encompass all disciplines taught on the semester. However, it is also wanted to clarify that had this project been ordered by an employer, it would have been implemented on just one of the devices. A typical choice could be to implement the system on the microcontroller. The microcontroller features peripherals implemented in physical hardware, for handling PWM and getting position and velocity readings from an encoder. As these are implemented in hardware in the microcontroller and since new data can be flagged by interrupts, eliminating the need for the microcontroller to poll the data, the microcontroller still has plenty of processing power left while handling all the tasks that the FPGA handles. The use

of the microcontroller as interface to the encoder would also eliminate the need for additional SPI functionality.

11 Conclusion

Using the ARTIX-7 FPGA and the TM4C123GH6PM microcontroller an implementation has been made to control the pan-tilt system. This implementation provides the required logic to facilitate PID-controllers to control the system.

A program containing tasks for handling data transmission via SPI, communication with a user through UART, and execution of the PID-controller algorithms has been developed and implemented on the microcontroller. The tasks are scheduled by the FreeRTOS operating system.

An average CPU utilisation of 46 % is determined satisfying the requirement that the PID-controllers are not to utilise more than 60 % of the total CPU time. An approximate overhead of 5.7 % is found and considering the CPU utilisation, it is deemed, that the overhead does not compromise the timing of the system. An UI task, utilising an UART driver handling the peripheral UART modules, has been developed to manage different user commands, for which a protocol has been designed.

Functionality has been developed to allow for communication between the microcontroller and the FPGA via the SPI protocol. Tests have shown messages transmitted via SPI to the FPGA, at a bit rate of 13.3 MHz, to be received correctly 10000/10000 times. This complies with the requirement of 99.9 % messages correctly received.

Functionality has been implemented on the FPGA to interface the motor encoders and keep track of the motor positions by counting the number of encoder counts relative to a reference position given by a reset routine implemented on the FPGA. Furthermore functionality to derive the velocity of the motors, based on the time between encoder counts, has been implemented. Obtained velocity measurements have been found to be erroneous, and the effects of this are discussed in the report.

To control the motors, functionality has been designed to produce a PWM-signal with a frequency of 48.8 kHz. The PWM-signal is dictated by a duty cycle reference received through SPI.

Mathematical models of the pan and tilt motors have been developed to allow for the systems to be simulated, and to be able to design PID-controllers using the pole placement method. Single position PID-controllers have been designed for both the pan and tilt motors based on the pole placement method. Furthermore, velocity PID-controllers have been designed for the tilt motor based on the pole placement method as well as the heuristic Ziegler-Nichols method. Lastly, using each of the two velocity controllers as the inner controller in a cascaded controller design, outer position controllers have been designed based on the pole placement method. In the design phase it proved somewhat difficult to thoroughly meet all the performance specifications, while still maintaining low enough gains that the required control signals were reasonable. The designed controllers have been implemented into the microcontroller and have been tested to evaluate the

performance and compare it to the performance predicted by the model. Here it was found that the tilt motor model seemed to be quite accurate, while the model of the pan motor is deemed inaccurate. Contrary to expectations, it was found that the cascaded controller design did not improve performance compared to the single controller design, while at the same time being more complex to design and tune. Tests showed that good performance of the inner controllers are crucial to a cascaded design. The Ziegler-Nichols design method gave significantly better results for designing the velocity controller compared to the pole placement based design. This may partly be caused by an erroneous velocity feedback, the effects of which, and means to correct it, have been discussed in the report. With an overshoot of 0.8 %, a rise time of 0.34 s and a settling time of 2.86 s for the tilt motor, the single position PID-controller using the pole placement method achieved the best result. The cascaded controller achieving the best performance, had an overshoot of 14 %, a rise time of 0.38 s and a settling time of 3.22 s using the Ziegler-Nichols method.

12 References

[1] Christoffer Sloth. *Projektbeskrivelse*. SDU. Can be seen in digital appendix.

[2] Robot technology 4th semester Group 04. *Flowchart Symbols*, 2019. Can be seen in digital appendix.

[3] robot electronics.co.uk/htm/emg30.htm. *EMG30 data*. Can be seen in digital appendix.

[4] ST. *DMOS FULL BRIDGE DRIVER*. Can be seen in digital appendix.

[5] Christoffer Sloth. Design of pid controllers. Lecture 4 in Control Engineering, 2019. Can be seen in digital appendix.

[6] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, chapter 8, Digital Control. Pearson, 7 edition, 2015.

[7] Chapter 5: Design of digital control systems using transform techniques, may 2019. Found in the digital appendix. Provided by Christoffer Sloth.

[8] University of Michigan. Control tutorials for matlab & simulink. http://ctms.engin.umich.edu/CTMS/index.php?example=MotorPosition&section=SystemModeling, may 2019.

[9] Mads Alber Kuhlmann-Jørgensen. Derivations and estimations for inertia, 2019. Can be seen in digital appendix.

[10] rexroth. Strut profile 45x45. https://www.boschrexroth.com/en/xc/products/product-groups/assembly-technology/basic-mechanic-elements/strut-profiles/strut-profiles-slot-10-modular-dimensions-45/45x45, may 2019.

[11] MISUMI. 8 series aluminum extrusions - overview. https://uk.misumi-ec.com/pdf/fa/2014/P2_0629-0630_F40_EN.pdf, may 2019.

[12] Tom S. Pedersen Palle Andersen. *Modeldannelse*, feb 2010. Danish. Can be seen in digital appendix.

[13] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, chapter 3, Time Domain Specifications. Pearson, 7 edition, 2015.

[14] Jacques Smuts. A tutorial on cascade control. http://blog.opticontrols.com/archives/105, may 2019.

[15] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, chapter 4.3, The Three-Term Controller: PID Control. Pearson, 7 edition, 2015.

13 Overview of Digital Appendix

1. **Figures**: Flowcharts, state diagrams and task diagram.
2. **Datasheets**: Datasheets for components.
3. **Articles**: Publications.
4. **Tests**: Data from tests.
   (a) Microcontroller timing test.
   (b) SPI test.
   (c) PID-controller tests.
5. **Calculations**: Additional calculations.
6. **Code**:
   (a) C code: libraries, drivers and source code developed.
   (b) FPGA code: Modules, top level.
7. **Minutes of meetings**
8. **Controller Design and Simulation**: Plots of all the controllers, MATLAB scripts.
9. **Video**: PID-controller demonstrated on pan-tilt system.
10. **Projektbeskrivelse** (Project description)

14 Word list

- **Pan-tilt system**: A robot with a motor to control a frame for panning and a motor for tilting.
- **Incremental encoder**: An encoder which does not keep track of the absolute position, but requires an external reference point.
- **Index signal**: A reference signal which provides a fixed reference point for the position.
- **Model**: A mathematical description of a system, e.g. electric motors.
- **Frame**: A set of aluminium extrusions bolted together which a motor can rotate. On the pan-tilt system there are two frames, referred to as the tilt-frame and pan-frame. They are similar but not identical.
- **Controller**: The control algorithm responsible for regulation of a given system.
- **Pole placement**: Method to place closed-loop poles of a plant in predetermined locations in the s-domain.
- **Saturation**: The point at which the the system cannot perform better due to physical limitations, e.g. a motor reaching its maximum angular velocity.
- **Single controller**: A controller design utilising only a single control loop as opposed to cascade control.
- **Cascade control**: A control design utilising several nested control loops.
- **Real-time**: The ability to guarantee that a task can initiate to a given time.
- **TM4C123GH6PMI**: The microcontroller used for the project.
- **Basys-3 Artix-7**: The FPGA used for the project. It is a development kit which contains the FPGA and peripherals like switches, LEDs, buttons, etc.
- **Overhead**: Excess computation time not used on running tasks.
- **Modules**: Self-contained software unit communicating through multiple input and output.
- **Motor encoder**: Used to describe the physical encoder on the motor.
- **Back emf**: Electromotive voltage which opposes the change of current which induced it.
- **MicroBlaze**: Soft microprocessor core, which is designed for Xilinx FPGAs.