Pan & Tilt

Regulering og kontrol af robotsystemer- Gruppe 5

Uddannelse og semester:

Robotteknologi - 4. semester

Dato for aflevering:

25-05-2016

Vejleder:

Sarah Madeleine Garney

Gruppe medlemmer:

Mathias Gregersen, Anton M.M. Schroll,

Daniel Haraldson, René T Haagensen,

Anders Ellinge, Bosse Hougesen

& Steen-Bjørn C.R. Vinther

Teknisk Fakultet

Syddansk Universitet

Preface

This report is written by 7 4th semester Robotteknologi students from University of Southern Denmark in spring 2016. The report is based on the 4th semester project. It is aimed at people with basic knowledge in the disciplines: Digital programmable electronics, Embedded programming, Control Engineering and Computer systems. It describes the theory and analysis involved in designing the controller used to control the Pan & Tilt system.
The report is written as a conclusion to the project and contains documentation of the process and the experiences gained. It should therefore be possible to recreate the project using this report.

1 Abstract

The purpose of this paper is to show how to make a controller and how to use it to control a spotlight consisting of two connected frames, two motors and one light. Precise motor control is an essential requirement in most applications involving any kind of actuators that needs to behave in a predictable and controlled way. A spotlight capable of relative precise movement is important when performing on a stage. In order to make a controller, several measurements of the system's response to different voltages have been analysed. Using Matlab and control theory the values of the controller have been estimated. The results show; with the right controller, the system is easily controllable and will behave in a predictable way.

Indholdsfortegnelse

2 Indledning

Indledning

3 Problem formulation

When performing on a stage, light in the right places is important. An easy way to get that, is via controllable spotlights. They have to be able to rotate horizontally and vertically, thereby targeting specific points on the stage and do so autonomously.
The Pan & Tilt System (P & T System) can be made to meet those requirements.

3.1 Definition of goals

The main goals of the product is, to be able to point the cone of light from the spotlight, at a specific point on a predefined surface or follow a human walking around on a surface. The user should have the option, to specify the limits of the spotlight's working space. The positions or paths should be set by the user and the movement should be pleasant to watch.

3.2 Requirements specification

The requirements will be divide into primary and secondary requirements, so as to identify the most important objectives.

3.2.1 Primary requirements

Hardware requirements

- The controller has to be implemented on the FPGA(Field-programmable gate array).
- The communication between the microprocessor and the FPGA has to be done with SPI(Serial Peripheral Interface Bus).
- The FPGA has to control the PWM(Pulse-width modulation) signal for the motors.

Driver requirements

- The system cannot have an overshoot of more than 10%.
- The system has to have a settling time of at least 1.5 sec for one revolution of either pan or tilt.
- At a scene of dimension 5 m x 5 m, from a hight of 5 meters, the system must achieve a velocity of at least $2.5m/s^1$ relative to the scene, which is the maximum velocity the average person would walk. [1]

Application requirements

- The application must support for the spotlight to be placed at different hight
- The application must support different stage dimensions up to 14 m x 14 m.

[1] Wikipedia, Preferred walking speed, https://en.wikipedia.org/wiki/Preferred_walking_speed, 23/05/16

- The application must needs to be able to point its light towards a specific position on a 14 m x 14 m stage at a hight of 5 meter, with a error margin of +/- 10cm.

3.2.2 Secondary requirements

- The movement between to points has to be smooth.
- The pan- and tilt system has to be able to be calibrated.
- The spotlight has to be able to be controlled by an application.
- The spotlight has to be able to be controlled by a joystick.

4 Indledende idé fase

Applikation
Hvordan kan arbejdsområdet afgrænses efter brugerens behov?
Hvilke input / parametre skal kontrolsystemet manipulere?
Controller
Hvordan kan controlleren modelleres?
Hvordan kan de forskellige opgaver opdeles i tasks?
Hvordan udarbejdes en styring til motoren?
Hvilke parametre ønskes kontrolleret?
Hardware
Hvordan anvendes de indbyggede sensorer?
Hvordan kan platformen modelleres?
Hvordan kan systemet styres pålideligt?
Hvilke fysiske begrænsninger medfører hardwaren?
Dette bygger på antagelsen at pan and tilt projektet altid virker
Bevæge systemet i forhold til koordinater (Systemet skal kunne bevæge sig, i forhold til givne koordinater)

5 The System

Here goes the tekst.

6 The Microcontroller

Given the design requirements in section 3.2, a block diagram was created for the entire system. This included all the functioality that is to be placed in the microcomtroller. From here the next stage is to define the content of these blocks, and how they interact with each other.

The microcomtroller is to establish an application programmer interface (API) to the P&T driver placed in the FPGA. The microcontroller will also run the applicaiton that uses this interface in order to help the user achieve his/her goal, which is to control the spotlight.

The program will be written in the C language, and it will be documented using task diagrams and states machines.

6.1 Choice of Operating System

Why use freeRTOS - The tool used in our semester - real tome system

what are the alternatives? there are my own operating system but the options here are limited

Witch limitations does FreeRTOS give os - no messages, but events - no binary semaphores - (handling floating point - whitch i have never tested)

6.2 Program Structure

Given the above mentioned requirements, a task diagram for the system was created, here the program were to handle two different sources of user input, in a manner that would not effect the application.

Insert task diagram here

walk through the system, separating the program into tasks starting from the block diagram. (see the criteria in Morten's ppt.)

one task one input queue

what drivers where required to achieve the goal of the program,

now add the used semaphores at critical sections

Now when the task diagram has been made, it is time to considerer possible structural weakness such as deadlocks and handling og queues.

6.3 The Application Programmer Interface

what is this sorcery - and what will i achieve by using it it is there to represent all the functionality available for each driver in the program.

where in the program will this be used

how will it be implemented

6.4 Event Queues and Shared State Memory

it will ensure that only events can be found in the queue, data will be saved and an event stating that data can be read will be send enstead

this is so that the data never will look like an event.

How is this implemented

6.5 Deadlock Protection

here three criterias neads to be held for a dead lock to take place

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait (add source XXXX)

All four conditions needs to hold true in order for a deadlock to occur. You can therefore provide a deadlock-free environment by avoiding one of the conditions.

Here we remove circular wait ...

Implementation of this idea ...

6.6 SPI-Master

using the build in free-scale spi-module in the TM4C123GH6PM Microcontroller, or creating the protocol from scratch.

The making of the state machine that is the SPI-master

A state is only a state if you have to wait till you move on

6.7 The Application

6.7.1 The Kernel task

making the state mashine for the kernel-task

separating commands in groups depending on the number of parameters to use.

the list of commands that will be supported (se the list on google drive XXXX)

6.7.2 The light-show task

making the state machine for the light-show-task

6.8 Overview

The above mentioned functionality was ment as a fully dressed system containing all the features that would be needed while working with a real spotlight. However, not all features was primary requirements, which makes it okay that it never got implemented.

The joystick: Even thought a driver was made for the joystick, it the the corresponding API was not. Therefore this

Point at a specific location.

This is the functions that did not made it to the implementation

1. ps2 controller and its corresponding api
2. calculating the tack-position from the relative coordinates given by the application
3. follow a waking person

7 The FPGA

Here goes the tekst.

8 Driver

This section describes how the driver is implemented in the FPGA. The section starts with an overall analysis of the system and afterwards goes into details with some blocks defined in the analysis and ends with an discussion on how the PID controller could have been implemented better as a filter.

8.1 Overall analysis

The easiest way to describe how the overall system works is by looking at the general components and how they are connected. The general components used in the implementation are: a comparator, a controller, a sensor, a motor driver with a PWM module and a SPI slave with a data bank. A generic multiplexer display was also added to simplify the debugging process, however this should not be seen as a part of the final product. Since the goal is to control two motors a separate controller setup was made for each motor. This means that each motor is connected to its own setup of a comparator, a sensor, a controller and a motor driver. The SPI slave component is common for the motors. On figure # Overall driver # an illustration of how the components are internally connected. This figure also shows information on the signals between the individual components.

SÆT BILLED AF OVERALL DRIVER IND

The components can be divided into the following blocks, each with a specific purpose.

The first block is the SPI slave block which contains only the SPI slave component. The purpose of this block is to receive and store information used by the rest of the components. It then shares the information with the components that are in need of them. An example of this could be the maximum and minimum speed of a motor which is send to the controller (this connection can also be seen on figure # Overall driver #).

The second block is the controller block which contains the comparator and the controller components. The purpose of this block is to regulate the motors according to a target input and the feedback. The block generates a duty cycle and a direction that is passed onto the motor driver.

The third block is the sensor block which consists only of the sensor component. The purpose of this block is to keep track of the current position of the motor. This is done by looking at the hall sensors from the motors and relating their output to the current position.

The last block is the motor driver which contain the motor driver component and its internal PWM module. The purpose of this block is to control and power the motor. This means that the block generates a PWM signal and outputs it on the correct pin according to the desired direction.

9 Motor driver block

This section will describe in detail how the motor driver block receives a duty cycle and a direction, then uses these to generate and output a **PWM** signal according to the direction and speed set by the controller block.

9.1 Description

An easy way to visualize the inner workings is to divide the Motor Driver into two separate blocks. The first block's task is to create a **PWM** signal which is equivalent to the duty cycle given by the controller. The second block's task is to set the output of the **PWM** signal to the right pin on the H-Bridge in order to control the direction of the motor.

9.2 Theory

This section will describe the theories used for the implementation.

9.2.1 PWM generation

The **PWM** signal is created based on a looping counter and a threshold. Imagine a counter resetting whenever it reaches a certain value, creating a loop. If the value is set to the period of the wanted **PWM** signal and the **PWM** signal output is dependent on the counter value relative to a threshold, then you will be able create a looping output that is dependant on the threshold. Figure # PWM implementaion principle # show an illustration of the behavior of **PWM** signal based on the looping counter and threshold.

NOGET MED H BRO

9.3 Implementation

The implementation was done with the separation of the block described before. A **PWM** component was created which has the sole purpose of creating a **PWM** signal based on a duty cycle. The other part has the purpose of directing the **PWM** signal to the right output pins based on the direction wanted.

9.3.1 PWM part

The PWM block was implemented so that whenever the counter value is below the threshold the PWM signal output is high and when the counter value is above the threshold the PWM signal output is low. The section of the PWM code that makes the counter can be seen on picture # Counting example #.

9.3.2 Direction part

This block is simply implemented as a switch that toggles on the wanted direction. As an example if the wanted direction is set to 0 then the PWM is driven to the relevant motor control output and the other is kept at zero.

9.4 Tests

To verify that this motor controller is working as intended, a test bench was created and various inputs was tested. The enable was set and a duty cycle of 8 bits was input, which corresponds to a certain duty cycle. A clear spike in PWM output could be seen, of the same length every time PWM is outputting. To further verify the test, the edges of the PWM signal was measured and used to find the exact duty cycle. Table # Table from other motor driver # showcases the tests of PWM.

9.5 Discussion

The tests show a deviation from the desired duty cycle of a small percentage. However these deviations have been deemed insignificant. The errors is created by the method that is used to calculate the PWM from the duty cycle, when converting the bit string, it is multiplied by the PWM period and divided by 256. This yields an integer, that when combined with the counter, yields a PWM very close to the desired duty cycle. Beyond that margin of error, a single clock cycle is lost due to resetting the counter, which corresponds to 1 ns.

9.6 Conclusion

The motor driver has been sufficiently explained and tested, and has become a solid module to create a PWM signal for the P & T system. The PWM calculation can be optimized very slightly by improving the conversion from duty cycle to PWM, and optimizing the counter, as to never lose any clock cycles.

10 The Controller

Here goes the tekst.

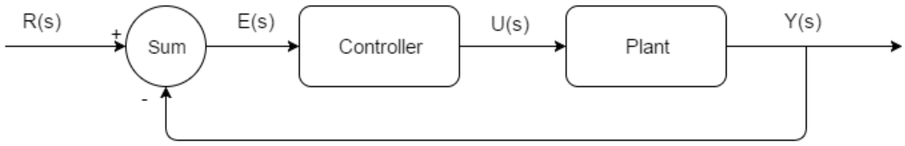
11 PID

11.1 Introduction

A controller is an essential part of most autonomous systems. This project is no exception - having control over where the pan and tilt system is pointing at is crucial to meeting the specified requirements. The following questions might then transpire.

- What controllers are available?
- Is a complex or simple controller desired?
  - Pros and cons?
- How do you determine which one to use?
- How do you implement them?

When discussing what controller to use, figure 1 is the point of reference, where the plant is the previously estimated transfer function, **SKRIV FORMLEN** EQUATION REF, of the motor of the system.

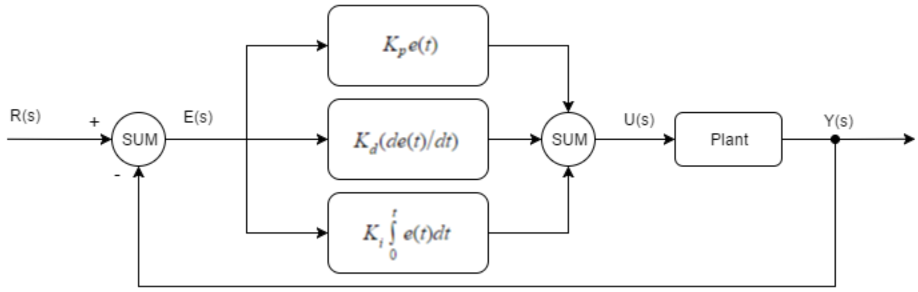


Figur 1: A generic way of representing a close-looped control system

11.2 The Controller

For simplicity's sakel Proportional (P), Proportional-Integral (PI), Proportional-Derivative (PD) and Proportional-Integral-Derivative (PID) be the only controllers taken into considerations for this project. Though Lead-Lag compensators could be considered as well, since in essence they can do the same as before mentioned controllers.

A PID controller is made up by three parts: the proportional gain looks at the current error, the integral looks at the past errors and the derivative looks at current rate of change. Each term has a tuneable gain called the k constants. In general these constants are what defines a PID controller's behavior. See figure 2 for illustration.



Figur 2: A representation of the PID controller principle

This is the equation of the standard PID controller in the time domain:
$K_p e(t) + K_i \int_0^t \mathrm{e}(t)\,\mathrm{d}t + K_d \frac{de(t)}{dt}$

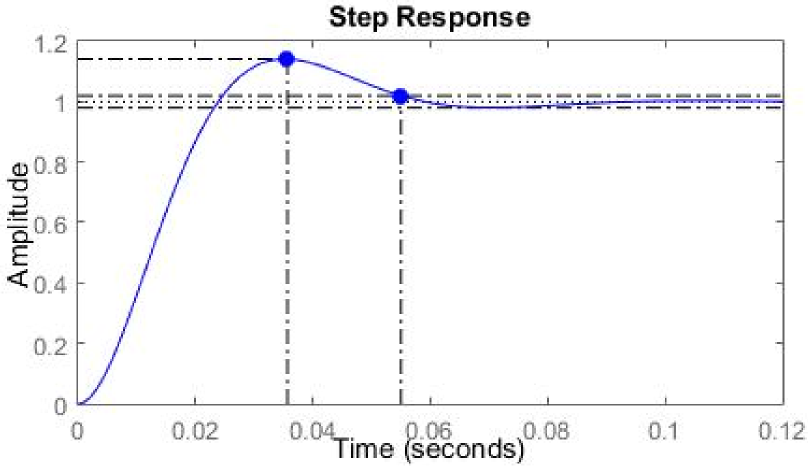
If transformed to the s domain the equations looks like this:
$G(s) = K_p + \frac{K_i}{s} + K_d s$

11.3 Proportional Controller

The P controller is by far the most straightforward controller, being that it is simply adding a gain to the open-loop transfer function. A P controller is a desired controller, since it is very simple to implement and tune, though this project have to meet the stated requirements.

To help analyse and give some intuition about how the motor's transfer function behaves, a root locus of the open-loop transfer function is plotted see figure 7. As one would expect it seems like this kind of controller will not suffice for the project.

As seen in figure 3, the step response of the closed-loop transfer function, has an overshot of 14.1 percent, and a settling time of 0.055 seconds. Meeting the specified requirements for the project with this controller is not going to happen. The overshot can be decreased by reducing the gain below 1, but this induces a longer settling time and vice versa.



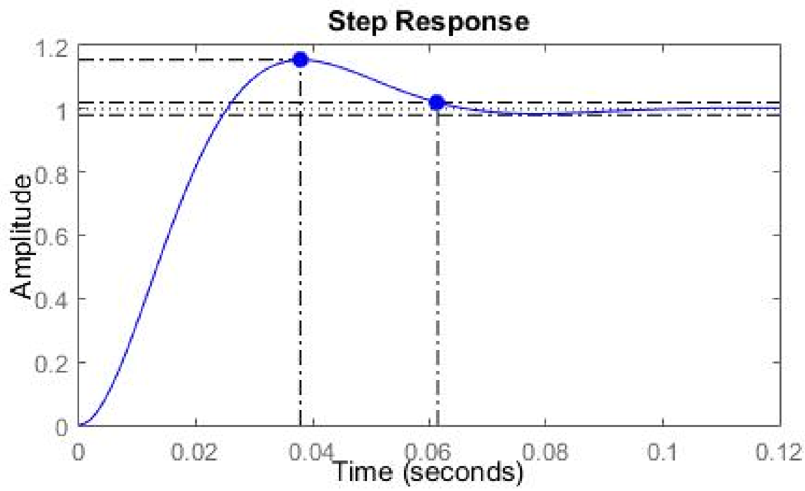
Figur 3: Peak amplitude: 1.14 - Overshot(%): 14.1 - At time(seconds): 0.0356 Settling time (seconds): 0.055

11.4 Proportional-Integral Controller

The PI controller accumulates past error terms over time to eliminate steady-state errors in the system. This can cause an increase in overshoot called the integrator wind up, and an increase in settling time. Tough if a requirement is complying a steady-state error of 0, then it is up to the designer to decide, if reaching state-state is more vital to the system, than having more overshot and a longer settling time.

Using a PI controller is the same as adding a pole and a zero to the open-loop transfer function. As seen in figure 7, this controller does not seem to live up to the requirements either. Placing the zero and pole differently could make a better controller, but it would never be able to adjust to both design criterias.

The step response of the PI controller is seen on figure 4, the only difference from the P controller is a slightly longer peak time and settling time and a larger overshoot, as expected. This controller would not meet the requirements.



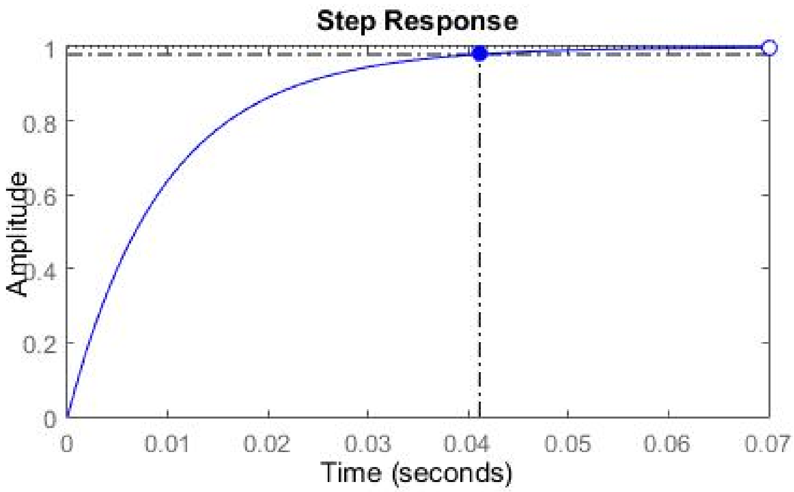
Figur 4: Peak amplitude: 1.15 - Overshot(%): 15.5 - At time(seconds): 0.0379 Settling time (seconds): 0.0614

11.5 Proportional-Derivative Controller

A PD controller is equivalent to the addition of a simple zero, **INSERT EQUATION HERE**, which improves the transient response. From a different point of view, the PD controller may also be used to improve the settling time or stability, because it anticipates large errors and attempts corrective action before they occur.

FIGURE5 shows a promising root locus of the PD controller, it seems like that with the right amount of gain the system would fulfill the requirements.

The step response on figure 5 shows, that in fact this controller would be ideal for this project. There is no overshot, and the settling time is well below 0.05 seconds.



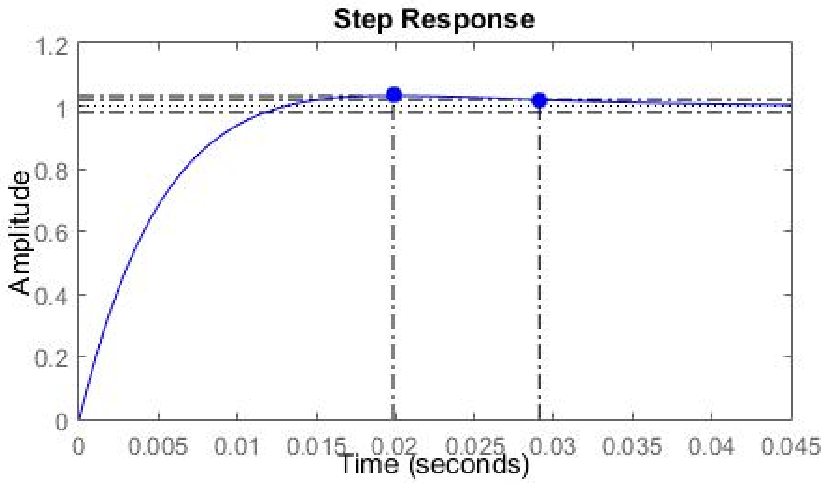
Figur 5: Peak amplitude: >= 0.998 - Overshot(%): 0 - At time(seconds): > 0.07 Settling time (seconds): 0.0411

11.6 Proportional-Integral-Derivative Controller

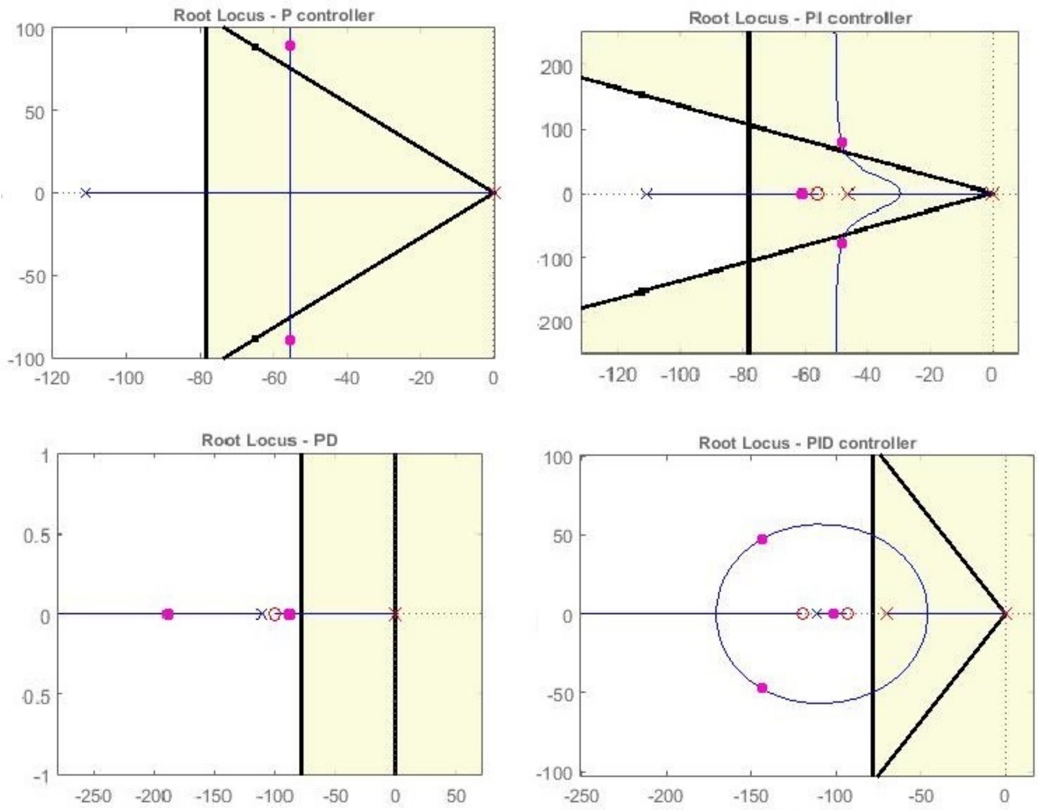
The PID controller is the most complex controller which is going to be discussed. As noted earlier an PD controller was sufficient for a controller. So why discuss the PID? The reason is that, in reality systems might have limitations which analysis have not taken into account yet. Reaching a steady-state error of 0 is actually a problem with a PD controller for this project, since the pan and tilt system does not function below a certain range of voltage, which means that an integral is needed.

Notice on figure 7, that an PID controller is the addition of 2 zeros and 1 pole to the open-loop transfer function. The root locus shows that this system with a relative gain still meets the requirements of the design.

The step response of the PID controller, figure 6, shows that indeed a PID controller would be a good choice of controller for this project. There is well below 10% overshot and the settling time is a great deal below 0.05 seconds.



Figur 6: Peak amplitude: 1.03 - Overshot(%): 2.92 - At time(seconds): 0.0201 Settling time (seconds): 0.0277



Figur 7: Open-loop transfer function Root Locus plots of the estimated motor transfer function, EQUATION REF, with different controllers. The shaded areas indicates designs not met if poles of the closed-loop transfer function is placed there. The designs are: Overshot > 10%, settling time ¡ 0.05 seconds. 7

12 Perspectivation

Here goes the tekst.

13 Conclusion

14 References

- Ref1
- Ref2
- Ref3
- Ref4

A Appendix

1. Code
2. Report
3. Documents