Helicopter - PID practical

Enzo Evers

October 2017

Contents

1 Introduction **2**

2 Materials and Methods **3**
2.1 Materials 4
2.1.1 Helicopter 4
2.1.2 Controller 4
2.1.3 Mosfet 4
2.1.4 LCD Display (16x2) 4
2.2 Methods 5
2.2.1 Design 5
2.2.2 Implementation 6

3 Implementation **8**
3.1 Transfer function 9
3.2 PWM 13
3.2.1 PWM reference and output pins 13
3.2.2 Mosfet rise time 15
3.2.3 Mosfet driver 19
3.3 PID 20

Chapter 1

Introduction

When flying a RC helicopter you may want to say that it needs be fly a certain amount of meter above the ground so that you don't get a fine when flying to high. The helicopter needs to adjust its motor speed so that it doesn't flies higher then the maximum height.

I want to make a prototype that demonstrates this behaviour. I take a toy RC helicopter and strip it down to only the main propellers and motors. This is mounted on a piece that goes over over two poles so that the helicopter can only go upwards. An ultrasonic sensor is mounted on the bottom to measure the height.

My prediction for the PID settings are:

- **P** will be the biggest value because I want the helicopter to get to the requested height as fast as possible.
- **I** will be the smallest value because it determines with how much force the error is adjusted. When experimenting in Matlab with high values for I, there was sometimes an overshoot.
- **D** will be the middle value because it accelerates the error handling in the beginning but slows the acceleration down when closing into the final value. When this value is big it accelerates very fast to almost the final value but takes a long time to get to the actual final value.

Chapter 2

Materials and Methods

2.1 Materials

2.1.1 Helicopter

I have a cheap RC toy helicopter that is stripped down to only the main motors and propellers. It was something like figure 2.1.

Figure 2.1: The type of helicopter of which the propellers and motors are used.

2.1.2 Controller

For the controller I will use an Arduino.

2.1.3 Mosfet

To get enough Ampere for the motors a logic level power mosfet is used. The mosfet I will be using is model IRL3202PBF.

2.1.4 LCD Display (16x2)

A display is used to show the current PID values and error between setpoint and actual value.

2.2 Methods

2.2.1 Design

Define the transfer function / plant

To fully specify the system, I need to know the transfer functions of the helicopter.

- Plant input : Voltage
- Plant output : Height is centimeters.
- System setpoint : Height is centimeters.
- Feedback : Unity feedback since the setpoint is of the same type and range as the output.

2.2.2 Implementation

Implement adjustable PID in Arduino

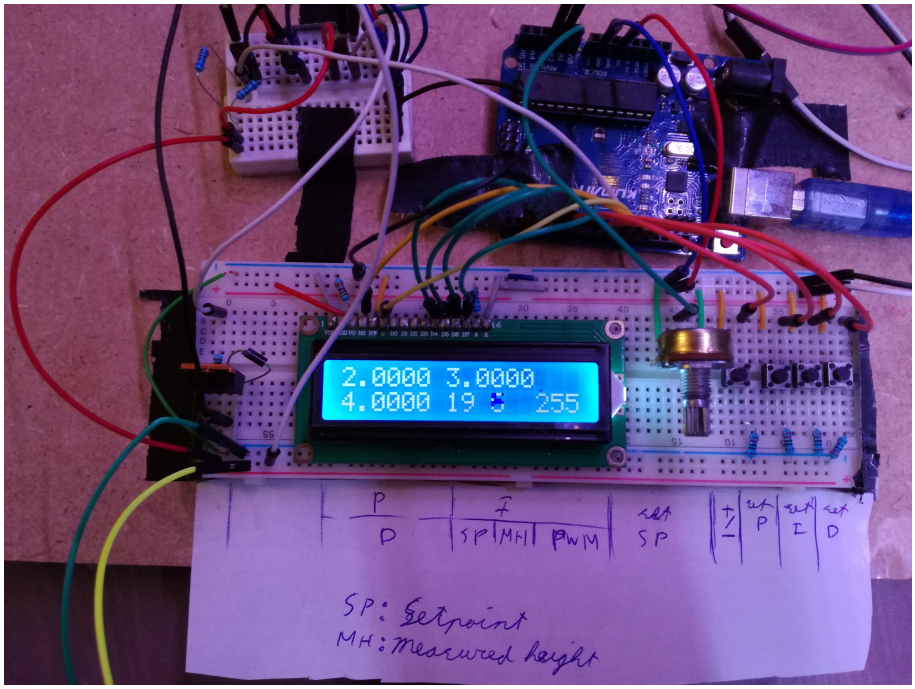

Figure 2.2: Caption

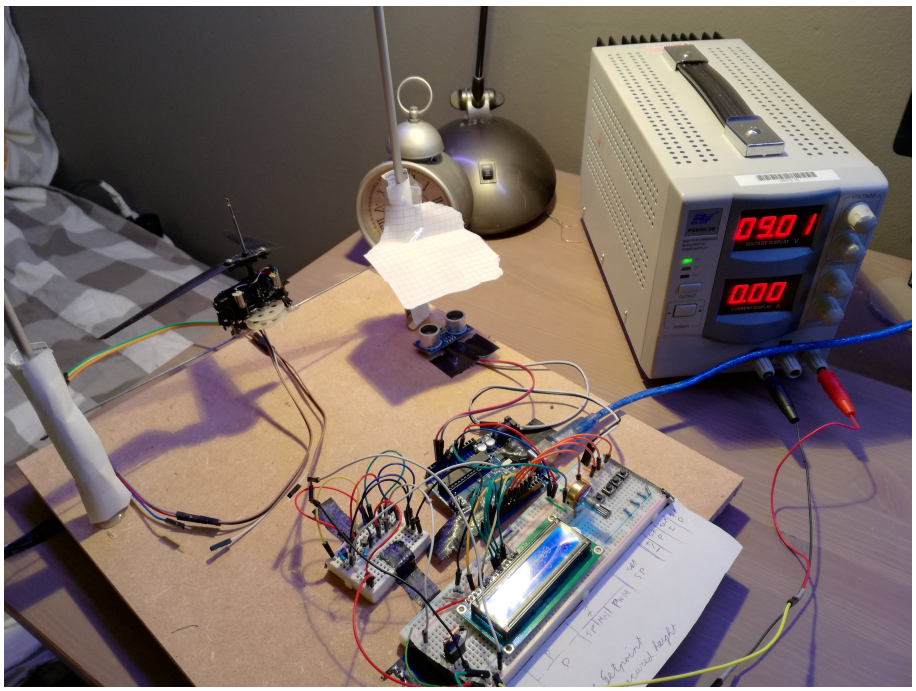

Figure 2.3: Caption

Chapter 3

Implementation

3.1 Transfer function

offset is 6cm

One thing that I concluded from the measurements is that the motors rapidly lose lift when their temperature rises.

Volt	Ampere	Height (centimeters)	Height - initial height (centimeters)	Rise Time (seconds)	Gain	Height increase
5	0.65	9	0	0	0	
5.2	0.69	9	0	0	0	0
5.4	0.72	9	0	5.5	0	0
5.6	0.75	9	0	5	0	0
5.8	0.79	9	0	4	0	0
6	0.82	9	0	2.6	0	0
6.2	0.87	10	1	3	0.16129	1
6.4	0.88	15	6	3	0.9375	5
6.6	0.92	18	9	3	1.363636	3
6.8	0.96	25	16	2.3	2.352941	7
7	1	30	21	2.8	3	5
7.2	1.02	45	36	2	5	15
7.4	1.05	47	38	2.5	5.135135	2
7.6	1.06	50	41	2.7	5.394737	3
7.8	1.08	64	55	3	7.051282	14
8	1.09	70	61	2.5	7.625	6
8.2	1	70	61	2.6	7.439024	0
8.4	1	80	71	3	8.452381	10
8.6	1	82	73	3	8.488372	2
8.8	1	87	78	2.8	8.863636	5
9	1	90	81	2.8	9	3
				2.733333	5.350996	5.4

Initial height 9

Figure 3.1: The gray rows indicate the rows of which the average is calculated below the columns.

A rough approximation of the transfer function of this system is defined with the following constants

- **Gain: 28.57**
  - The most stable part of is from 6.2 volts to 9 volts. From the data table in figure 3.1 you can see that there is an exponential gain. The derivative of the range between $\frac{81cm-1cm}{9V-6.2V} = \frac{80cm}{2.8V} = 28.57$ and this is $\frac{28.57}{5} = 5.6$. So on average the helicopter rises 5.6cm per 0.2V. Looking at the average increase in figure 3.1 this seems about right. The gain is 28.57. One important thing about this gain is that 6 volts nee to be subtracted from the input voltage for the first order transfer function to work. From 6 volts on the the helicopter starts to get lift.
- **Settle time: 2.733 seconds**
  - Taking the average of the rise times of the measured points between 6.2 and 9 gives an average settle time of around 2.733 seconds.
- **Time constant: 0.68 seconds**
  - The time constant is the settle time divided by four.

The resulting approximation of a first order transfer function will look like this:

$$H(s) = \frac{28.57}{0.68s + 1} \tag{3.1}$$

Simulating this in Matlab gives roughly the right output.

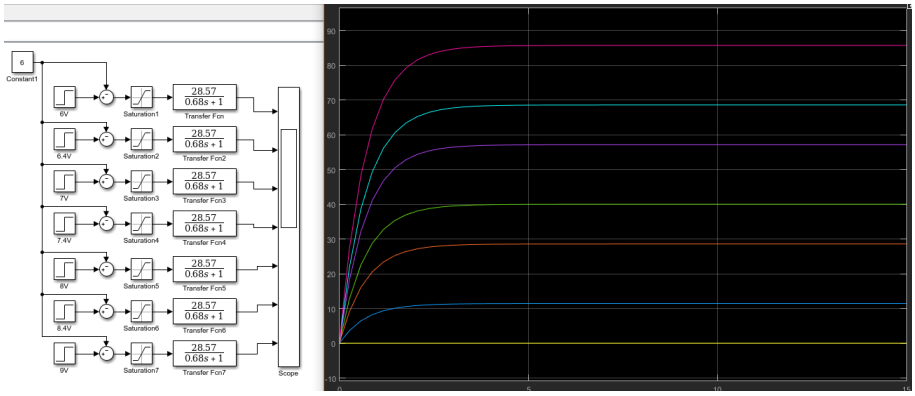

Figure 3.2: Caption

When adding a PID controller with no filled in values the transfer function becomes this.

$$H(s) = \frac{28.57K_Ds^2 + 28.57K_Ps + 28.57K_I}{s^2(0.68 + K_D) + s(1 + K_P) + K_I} \tag{3.2}$$

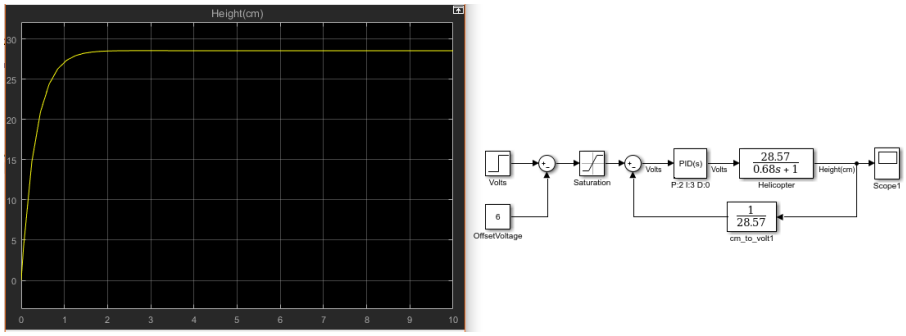

Figure 3.3: Caption

The systems can be modeled using either voltage or height(cm) as setpoint.

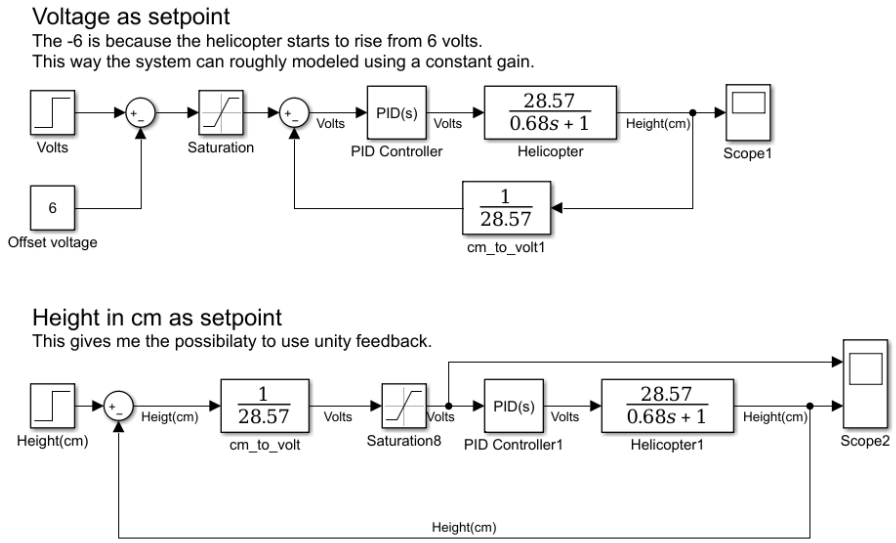

Figure 3.4: Caption

Considering the user input (potmeter controlling height) the system that uses height cm as setpoint seems like the logical one to choose. The other benefit is that the feedback now becomes unity feedback.

When taking the height as setpoint the final transfer functions becomes:

$$H(s) = \frac{K_D s^2 + K_P s + K_I}{s^2(0.68 + K_D) + s(1 + K_P) + K_I} \tag{3.3}$$

3.2 PWM

The original control board in de helicopter can be seen in figure 3.5.

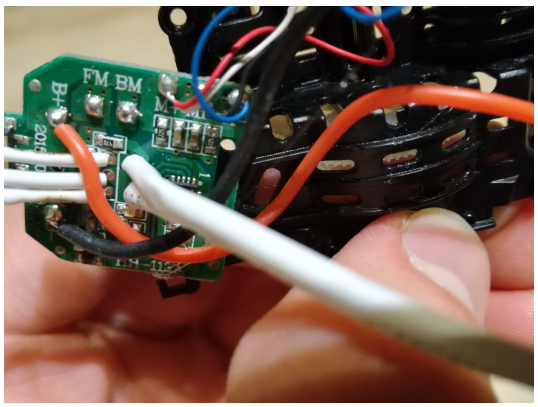

Figure 3.5

3.2.1 PWM reference and output pins

To get a better feeling of the difference between the desired PWM signal and the actual PWN signal over the mosfet I use pins 10 (OC1B) and 9 (OC1A) of the Arduino uno [1]. The reason why I use these is because they are outputs for the same timer. Figure 3.6 shows the part of the datasheet where this is stated.

- **$\overline{\text{SS}}$/OC1B/PCINT2 – Port B, Bit 2**

$\overline{\text{SS}}$: Slave Select input. When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB2. As a Slave, the SPI is activated when this pin is driven low. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB2. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB2 bit.

OC1B, Output Compare Match output: The PB2 pin can serve as an external output for the Timer/Counter1 Compare Match B. The PB2 pin has to be configured as an output (DDB2 set (one)) to serve this function. The OC1B pin is also the output pin for the PWM mode timer function.

PCINT2: Pin Change Interrupt source 2. The PB2 pin can serve as an external interrupt source.

- **OC1A/PCINT1 – Port B, Bit 1**

OC1A, Output Compare Match output: The PB1 pin can serve as an external output for the Timer/Counter1 Compare Match A. The PB1 pin has to be configured as an output (DDB1 set (one)) to serve this function. The OC1A pin is also the output pin for the PWM mode timer function.

PCINT1: Pin Change Interrupt source 1. The PB1 pin can serve as an external interrupt source.

Figure 3.6: From the atmega328p manual [1]

I compared the delay between pin 10 and 9 and between 10 and 11. These are all PWM pins but pin 11 uses timer 2 for it's PWM while pin 10 and 9 use timer 1. Pin 9 reacts almost 75 times as fast as pin 11 when comparing to pin 10.

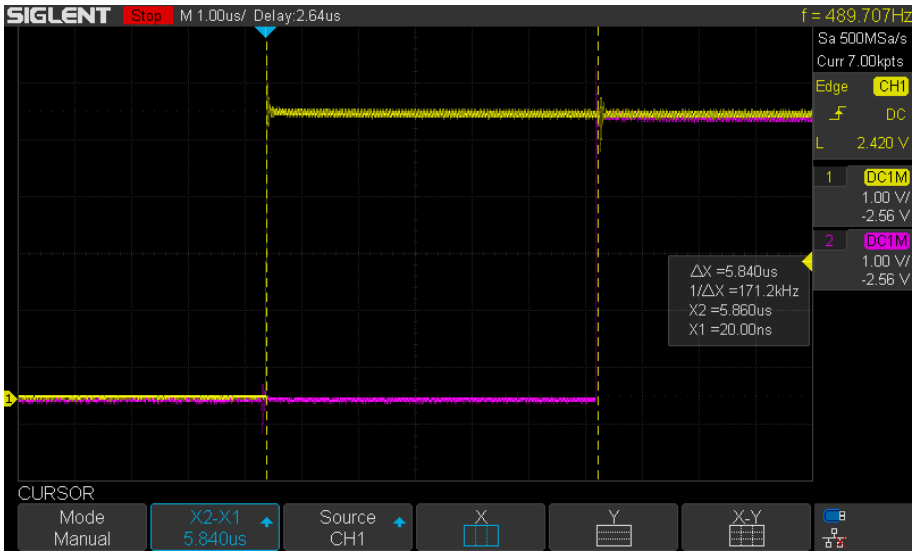

Figure 3.7: Yellow: pin 10. Purple: pin 11. Delay: 5.840us = 5840ns.

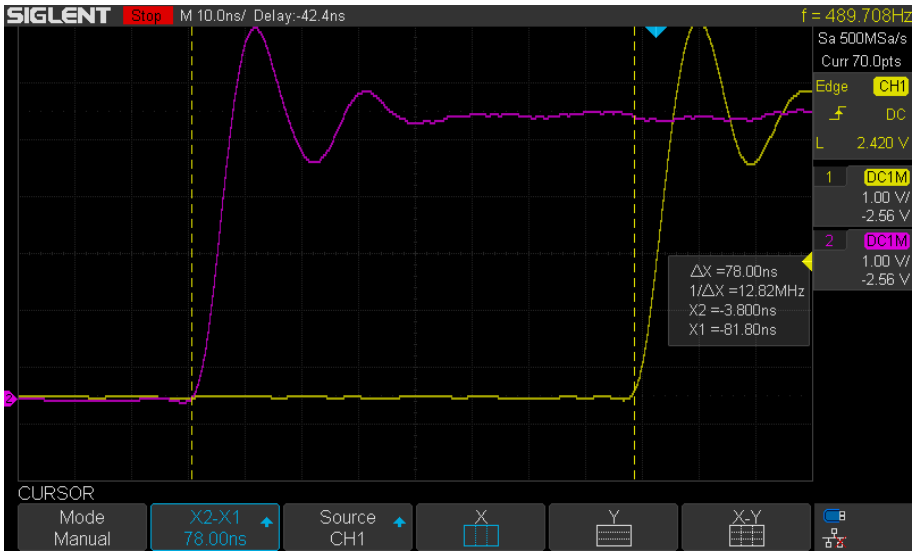

Figure 3.8: Yellow: pin 10. Purple: pin 9. Delay: 78.00ns = 0.078us.

3.2.2 Mosfet rise time

Now I will test how much time the mosfet takes to go from 0 to 5 volts. The first mosfet I tried was a IRF640. It turned out that this mosfet was to slow. I got the advice to use a logic level mosfet instead. The IRL3202. These types of mosfets have a lower $V_{GS(th)}$ and V_{GS} which results in faster switching time.

It was a surprise to that the IRF640 had a faster rise time then the logic level mosfet. The difference between the first time I tested the mosfet and the second time I tested the mosfet is that for the first time I didn't use an external body diode and the resistor was of a higher value. A higher resistor value ofcourse increases the rise time but didn't think that it would be faster then a logic level mosfet.

Gate resistor	10Ω
External body diode	
PWM duty cycle	50%
PWM voltage	5V
Load voltage	9V
Mosfet variant 1	IRF640
Mosfet variant 2	IRL3202

Mosfet variant 1: IRF640N

Without mosfet driver, power supply off

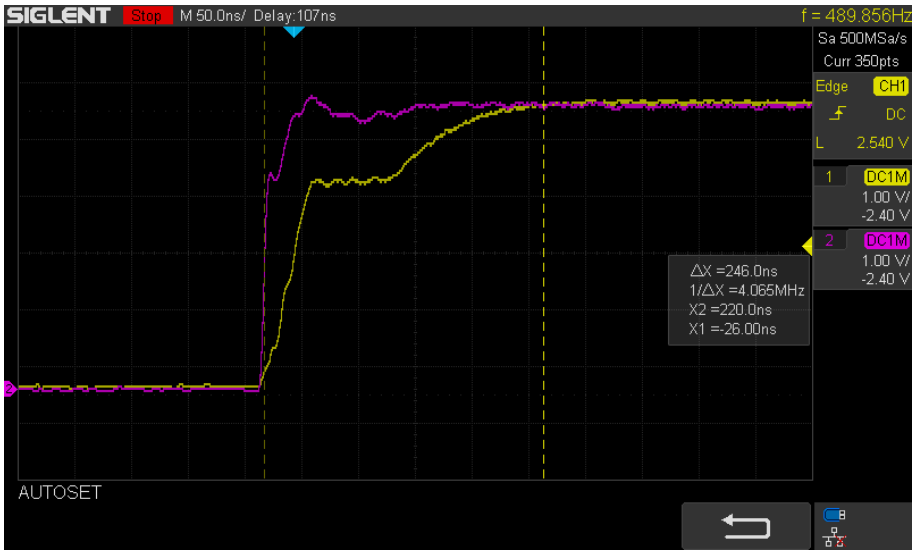

Figure 3.9: Without mosfet driver, power supply off, Yellow: Gate, Purple: ref

Without mosfet driver, power supply on

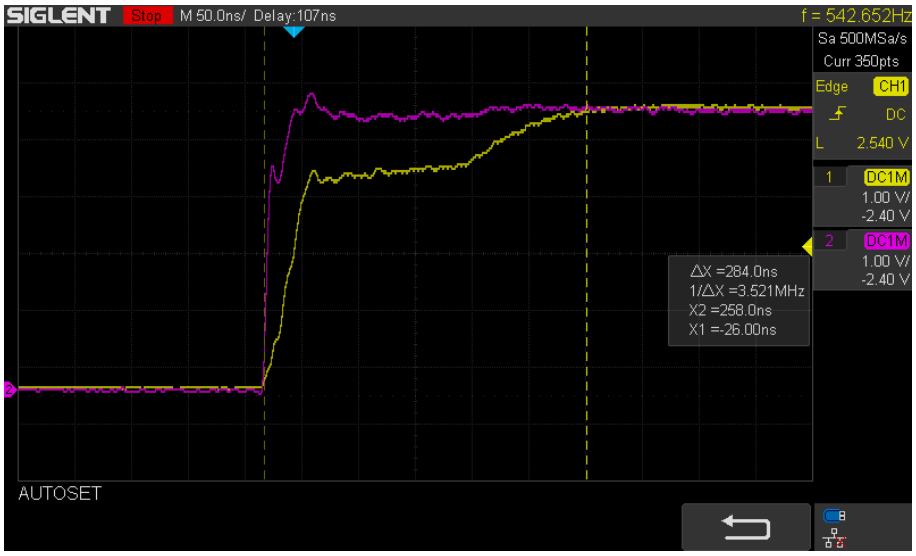

Figure 3.10: Without mosfet driver, power supply on, Yellow: Gate, Purple: ref

With mosfet driver, power supply off

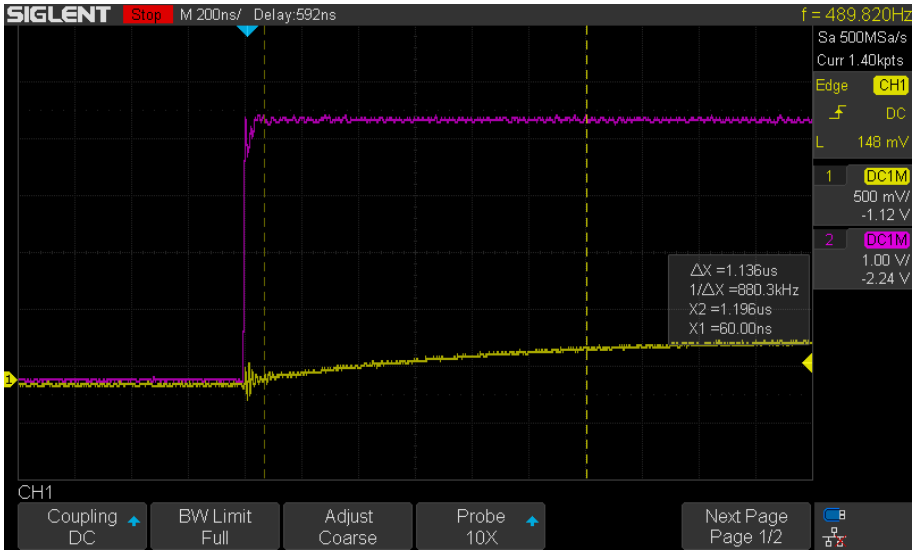

Figure 3.11: With mosfet driver, power supply off, Yellow: Gate, Purple: ref

With mosfet driver, power supply on

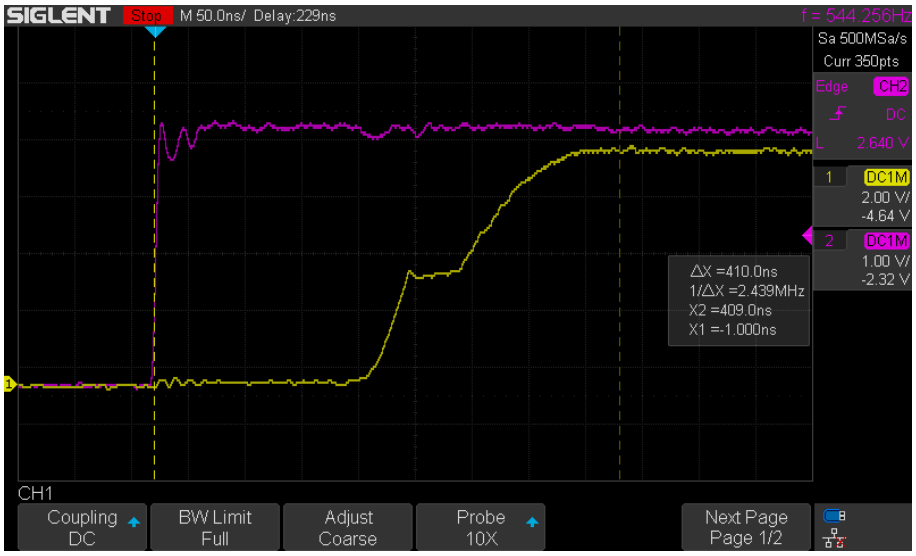

Figure 3.12: With mosfet driver, power supply on, Yellow: Gate, Purple: ref

Mosfet variant 2: IRL3202

Without mosfet driver, power supply off

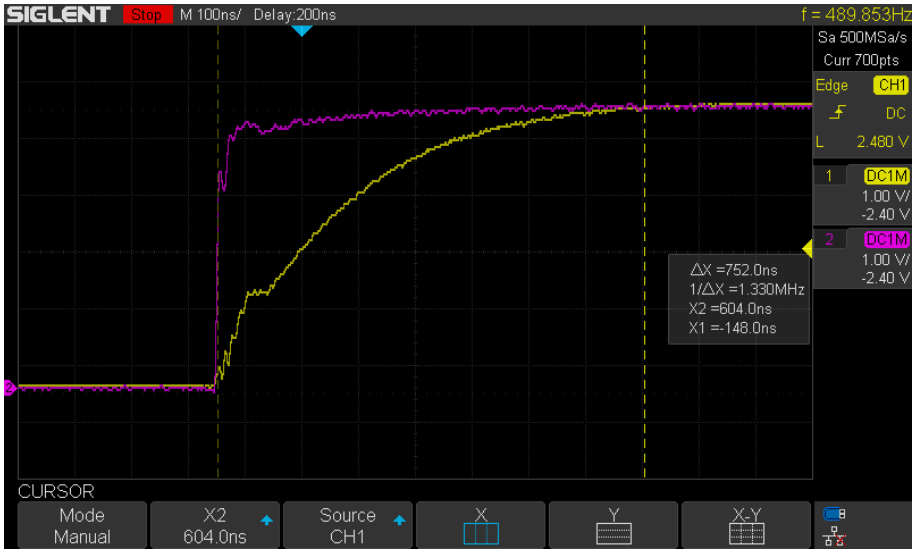

Figure 3.13: Without mosfet driver, power supply off, Yellow: Gate, Purple: ref

Without mosfet driver, power supply on

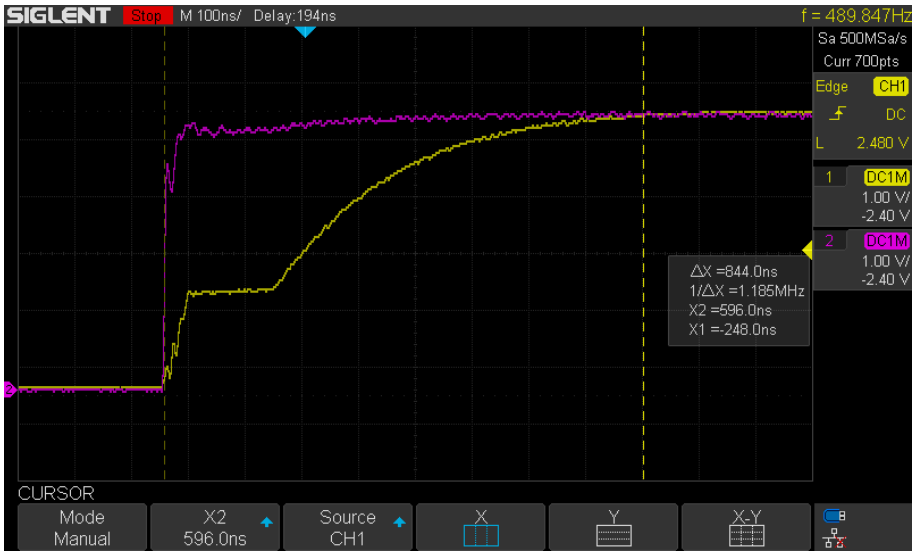

Figure 3.14: Without mosfet driver, power supply on, Yellow: Gate, Purple: ref

With mosfet driver, power supply off

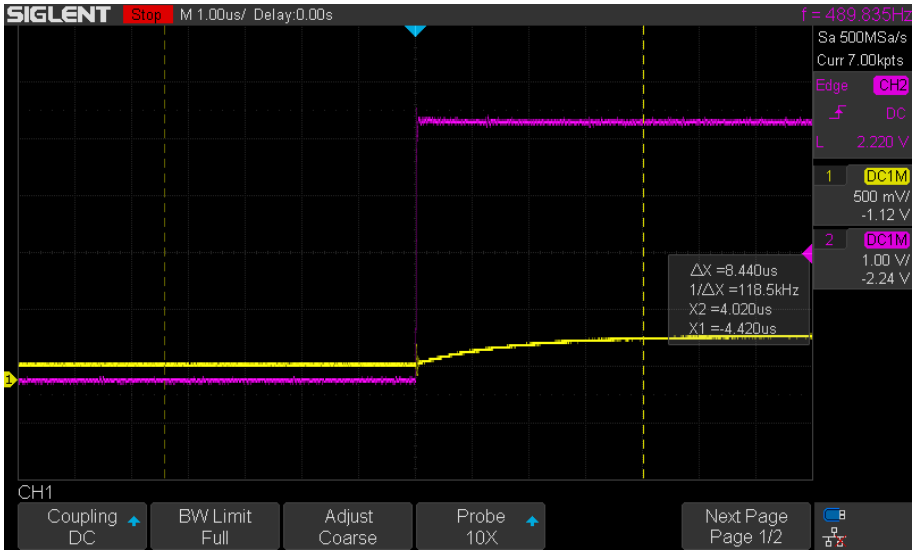

Figure 3.15: With mosfet driver, power supply off, Yellow: Gate, Purple: ref

With mosfet driver, power supply on

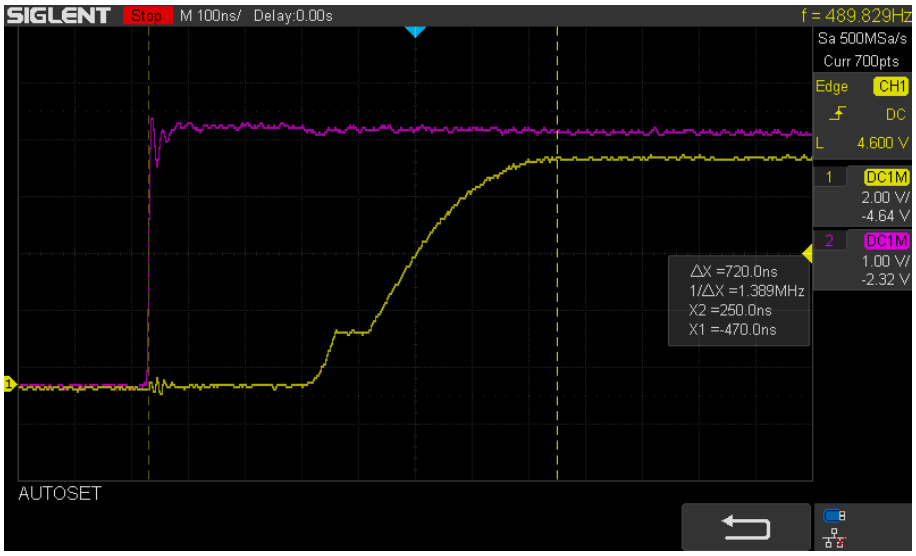

Figure 3.16: With mosfet driver, power supply on, Yellow: Gate, Purple: ref

3.2.3 Mosfet driver

For the mosfet driver I used three S8050 NPN transistors and one S8550 PNP transistor. I chose these because the datasheets states that these are designed for class B push-pull audio amplifiers and that these two NPN and PNP transistors complement each other.

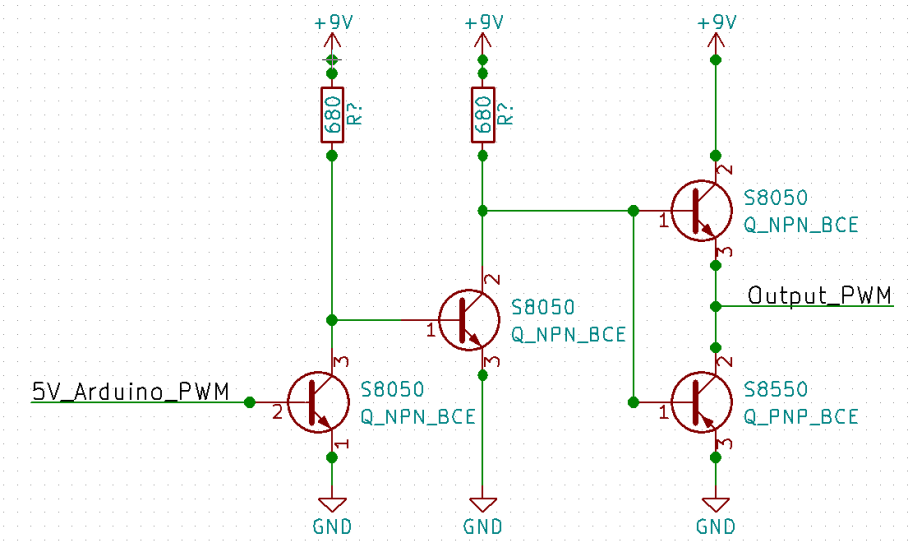

Figure 3.17: Caption

3.3 PID

For implementing the PID in code I read a blog on http://brettbeauregard.com [2] witch explains implementing digital PID quite well.

When applying the PID on the real helicopter you encounter some real world problems. The helicopter guiders (paper tubes around the aluminum pipes) were 'scraping' against the pipes because the helicopter went a bit sideways. An other bit issue was noise is the distance sensor. The noise could be filtered for some part by applying a moving average and boundaries for values changes. But there was still noise.

A video demo can be watched via this link: https://youtu.be/MnJwhAkHdoI.

The reason why it doesn't goes up that smooth is because the noise in the distance sensor messes with the Integral and Derivative parts of the PID. The Derivative part acts on change in measurement. When the measurements go up and down really quickly the slope of the measurements will be very steep. The Integral part is also affected by the noise because the sum of all the errors will go up and down quickly. For the integral part however this is not as big of a problem as it is for the derivative part since the noise is a relative small portion of the whole sum of errors.

Bibliography

[1] Atmel. *ATMEL 8-BIT MICROCONTROLLER WITH 4/8/16/32KBYTES IN-SYSTEM PROGRAMMABLE FLASH*. URL: http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf.

[2] brettbeauregard. *Improving the Beginner's PID – Introduction*. URL: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/.