

Analog PID module

Yves Stauffer, Microengineering

Assistant 1: Samir Bouabdallah

Assistant 2: Daniel Burnier

Professor: Roland Siegwart

Flying robots are an increasing center of interest. In order to achieve success in this novel domain control has to be perfectly mastered. In this context the ASL wanted to develop a fully analog motor PID (Proportional Integral Derivative) controller. Initially FPAA was supposed to be used, but it turned out that this fairly recent technology was not yet ready for such challenging tasks.

The chosen design is composed of two independent modules.

Command module:

Is receiving a digital (I2C) speed command and outputs an analog equivalent speed command. This module also provides a stable $\pm 12V$ power source from 5V. There is a triangle wave generator that will be used later to create the PWM. Finally, it is able to drive four different PID modules.

PID module:

Is performing the PID control, the PWM generation and the motor powering. Then it also reads and filters the motor's speed.

Note that the two modules have been built in a way that all the values ($K_{p,i,d}$, the filter gain, ...) can be changed and tuned easily. Furthermore there is a possibility to work in open-loop as well. And one can also operate the H-bridge (used to drive the motor) in two different modes. In one mode the possibility of reversing the motor's direction of rotation is disabled (i.e. only breaking).

Tests:

Both modules were tested thoroughly before a fully operational closed loop PID control was done.

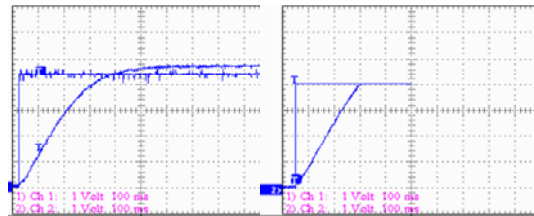


Figure 1 : open (left) and closed loop response (right)

The left part of figure 1 is the open-loop response. The motor is commanded to go from 0 rpm to 3500 rpm. The step signal is the command and the curve is the motors speed converted in equivalent voltage. Observe the steady state error.

The right part of figure 1 is the same experiment performed in closed loop with “optimal” PID values. Observe that the steady-state goes to zero, however the rise time is slightly the same.

Simulations of both cases were done, and exactly the same curves were obtained. Thus the command module and PID module are working correctly.

Conclusion:

It was shown that building and using a fully analog PID controller was feasible. Further investigations could lead to an upgrade to FPAA technology.

Table of content

1 INTRODUCTION	7
1.1 My project	7
1.2 How this report is organized	8
1.3 PID control : short overview	8
1.4 FPAA?	9
2 THE DESIGN.....	11
2.1 Goals.....	11
2.2 PID and FPAA?	11
2.3 The chosen solution	12
2.4 Advantages of this solution.....	14
2.5 Other possible solutions	14
3 THE CENTRAL MODULE: THEORY	15
3.1 5V -> $\pm 12V$	15
3.2 DAC.....	15
3.3 Triangle wave.....	16
3.4 Voltage follower	18
3.5 Summary of the used components	19
4 THE PID MODULE: THEORY.....	21
4.1 Error = command – feedback	21
4.2 PID.....	21
4.3 Inverter	23
4.4 PWM	23
4.5 H-Bridge	25
4.6 Motor.....	26
4.7 Speed encoder	26
4.8 Frequency doubling.....	27
4.9 Frequency to voltage	27
4.10 Butterworth filter	29
Analog PID module	3

4.11 Open loop control	30
4.12 Fuse	31
4.13 Summary of the used components	31
5 DESIGNING THE CONTROLLER ($K_{P,I,D}$)	33
5.1 Simulation model for matlab	33
5.2 Creating the simulink function	34
5.3 Interfacing with Simulink	34
5.4 Complete simulink model	34
5.5 Steady state transfer function (open loop)	35
5.6 Steady state transfer function (closed loop)	36
5.7 Conversion table	37
5.7.1 Closed-loop	37
5.7.2 Open-loop	38
6 CENTRAL MODULE: TESTS	39
6.1 5 -> $\pm 12V$	39
6.2 DAC	39
6.2.1 Using a Hirschman board	39
6.2.2 On the PCB	39
6.3 Triangle wave	40
6.4 Consumption	41
7 PID MODULE : TESTS	43
7.1 Error = command - feedback	43
7.2 PID	43
7.3 PWM	43
7.4 H-bridge	44
7.5 Motor	45
7.6 Speed encoder	45
7.7 Frequency doubler	46
7.8 Frequency to voltage	47
7.8.1 Using a Hirschman board	47
7.8.2 On the PCB	48
7.9 Butterworth filter	50
7.9.1 Using a Hirschman board	50
7.9.2 On the PCB	51
Analog PID module	4

7.10 Consumption	52
8 OPEN LOOP (WITHOUT PID).....	53
9 CLOSED LOOP: K_P ONLY	55
9.1 error = command – speed ($K_p = 1$).....	55
9.2 $K_p = 6$	56
10 CLOSED LOOP WITH PID	59
10.1 Non optimal PID values	59
10.2 Better values from Simulink	60
10.3 Overshoot.....	61
10.3.1 Reverse disabled	61
10.3.2 Reverse enabled.....	62
10.4 Optimal PID values	64
10.5 Tachymeter test	64
11 CONCLUSION.....	67
11.1 General conclusion	67
11.2 Are the goals met?	67
11.3 What can be improved	67
11.4 Special thanks to	68
12 BIBLIOGRAPHY	68
13 ANNEXES	69
13.1 Protel schematics	69
13.1.1 Original Central PCB	69
13.1.2 New Central PCB	69
13.1.3 Original PID PCB.....	69
13.1.4 New PID PCB.....	69
13.2 ACORT files.....	69
13.2.1 Central PCB files	69
13.2.2 PID PCB files	69
13.3 Relevant data sheets (extracts).....	69
13.3.1 DAC: MAX 520	69
13.3.2 H-Bridge: A 3949	69
13.3.3 Frequency to Voltage converter: LM 2907.....	69
13.3.4 Motor	69
13.3.5 Speed encoder.....	69
13.3.6 Reductor.....	69
13.4 Mathematica code.....	71
Analog PID module	5

13.5 PCB pictures	71
13.5.1 Central PCB front	71
13.5.2 Central PCB back	73
13.5.3 PID PCB front	75
13.5.4) PID PCB back	77

1 Introduction

Over the past decade flying robots have been an increasing field of studies. At EPFL (Ecole Polytechnique Fédérale de Lausanne) and more specifically ASL (Autonomous System Lab) this field is being tackled by several people.

Two main projects are currently going on. The first one is Adaptive Vision-based Flying Robots, which includes the blimp and slow flyers. The second is VTOL (Vertical Take-off and Landing), which includes the “flying Alice” and OS4 (Omnidirectional Stationary Flying Outstretched Robot, see figure 1).



Figure 1 : OS4

Flying robots are challenging on several levels, the two most important aspects being **lightness** and **control**. The latter is even more critical for helicopter like robots such as the OS4 because of the natural instability of the helicopter.

1.1 My project

One has to be aware that « control » is the key to most technologies. While robots need to have it implemented in hardware (or software), the human brain does it « by itself ». Indeed, when one is driving and has to turn right, the way one turns the steering wheel will be different if the curve is important or not as it will be different if one is driving at 50km/h or 120km/h on a highway. Several kinds of controllers exist, P (for Proportional), I (for Integral) and D (for Derivative) and its combination (PI, PD, ...). The most commonly used controller is PID. For more theoretical explanations please refer to section 1.3: “ PID control: short overview”.

The ASL has developed a standard PID module, which is used in various (even not flying) robots. The core of this module is based around a PIC microprocessor.

It is important to know that at the beginning all controllers were implemented in hardware using OA (Operational Amplifiers). Unfortunately this didn't allow an “on the fly” reconfiguration, in other words, once the values for $K_{p,i,d}$ were set they could only be changed by switching the values of capacitors or resistors.

Recently FPAA (Field Programmable Arrays) allowed to control dynamically (in some cases) values of capacitors and/or resistors and thus change $K_{p,i,d}$ without having to desolder anything.

Of course this is of great interest for applications that require a fast response time as well as a zero failure rate, this is why ASL proposed as a semester project to develop a PID controller based on this new and promising FPAA technology.

Analog PID module

1.2 How this report is organized

Chapter 1: Introduction

Chapter 2: The chosen design

Chapter 3: Theory on the central PCB (Printed Circuit Board), the one with the DAC

Chapter 4: Theory on the PID PCB

Chapter 5: Designing the controller, getting the values for $K_{p,i,d}$

Chapter 6: Test of the central PCB

Chapter 7: Test of the PID PCB

Chapter 8: Open-loop tests

Chapter 9: Closed-loop tests with K_p only

Chapter 10: Closed-loop tests with PID

Chapter 11: Conclusion

Chapter 12: Bibliography

Chapter 13: Annexes

1.3 PID control : short overview

Doing a complete course on controllers is beyond the scope of this report. If needed the reader can report to [1] Control Systems by Niese or [2] Feedback Control Systems by Phillips and Harbor.

In most problems we are given a Plant (motor for instance coupled with gears, etc) and have to determine what controller has to be chosen in order to meet the goals (settling time, overshoot, rise time, ...). A common closed loop system is depicted on figure 1.3a.

The following vocabulary is needed:

System input: user selected value that determines the output (i.e. speed)

Error: is the input minus the actual value of the considered variable

Manipulated variable: output of the controller

System output: output of the considered system (i.e. shaft speed)

Sensor: device that measures the output

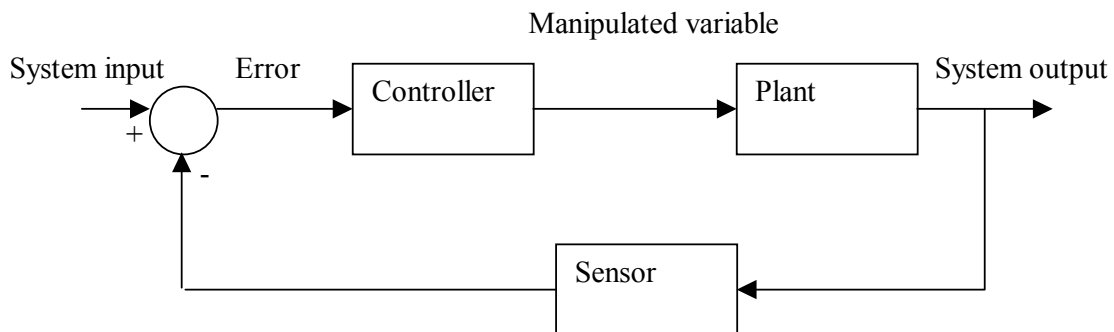


Figure 1.3a : typical closed loop controller

Note: open loop theory won't be considered in this chapter. It is a trivial case where the command is directly connected to the plant and no feedback is provided. However this function has been implemented on the hardware design as well.

In a common design the only “variable” is the controller (the plant and the sensors are usually given). Fortunately a great variety of controllers has been developed and each case is suited for a different one. Just to name a few: lag, lead, lag-lead, P, I, D and all the possible combinations of P,I and D.

As a matter of fact the most used controller is called PID (Proportional, Integral and Derivative). This is the case because it regroups the advantage of P, I and D controllers. By setting the right values to K_p , K_i and K_d one can achieve a good response. Refer to figure 1.3b for more information.

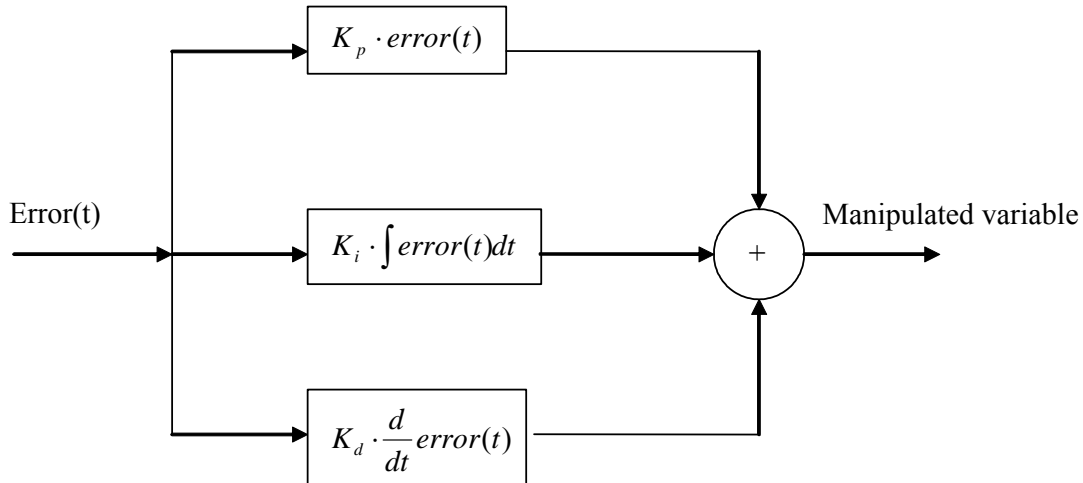


Figure 1.3.b : PID controller

Recall that:

K_p: compares output with system input

K_i: eliminates the steady state offset

K_d: anticipates changes by using the derivative, prevents the error of getting too big

1.4 FPAA?

FPGA stands for Field Programmable Gate Arrays and are the digital counter part of Field Programmable Analogue Arrays. While the former is already a well established and used technology the latter is fairly young. As a matter of fact no one used the analog capabilities of these circuits at EPFL.

Several companies are providing different FPAA chips. The ones considered were Anadigm, Lattice and Cypress.

Explaining how an FPAA works is beyond the scope of this report. One simply has to recall the general idea. Most FPAA contain a certain number of building blocks, usually made of operational amplifiers (OA), and switched capacitors (i.e. capacitors which can change their value). The user can play on several levels: first the inputs and outputs can be selected, second the connectivity between the different OA can be changed, third the value of the capacitors.

On figure 1.4a one has an overview of Anadigm’s chips, on the left one can observe the inputs. Figure 1.4b focuses on the CAB (Configurable Analog Bloc), which can be viewed as building blocks, note the presence of amplifiers and switched capacitors.

Analog PID module

Companies such as Anadigm provide a user friendly design tool which does the linking between the OA and allows a pretty fast evolution. Cypress on the other hand asks the user to “root” the layout manually, which offers more flexibility but also requires more knowledge and time.

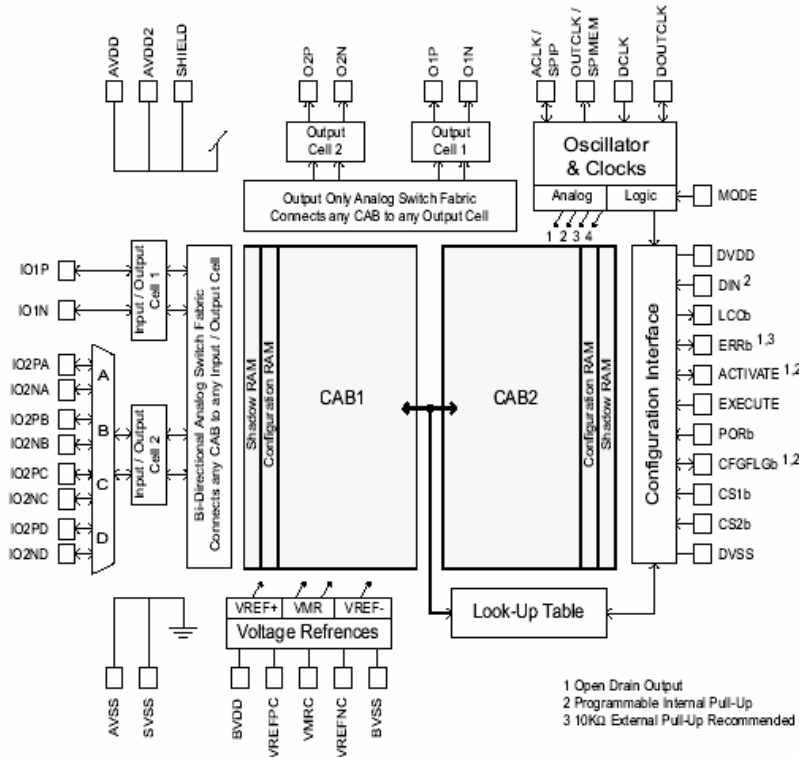


Figure 1.4a : Anadigm’s FPAA chip

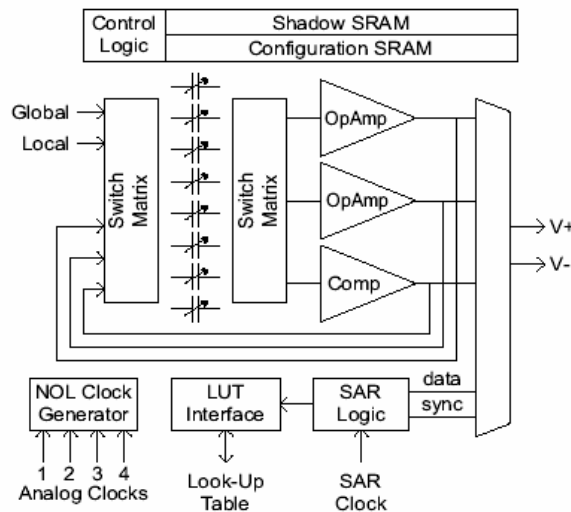


Figure 1.4b : CAB

2 The design

In this section one will find out what the specific goals of this project were. Indeed building an analog PID module is not a precise specification.

2.1 Goals

The goal is to design an analog PID module, it's application is aimed towards the OS4. This sets some crucial design aspects.

- 1) Speed: we want a FAST control
- 2) Lightness: the whole module has to be as light as possible
- 3) The OS4 is composed of 4 motors, thus 4 PID controllers are needed
- 4) Communication: RS232 and/or I2C
- 5) Command by PWM (Pulse Width Modulation) a motor that would consume around 1-2 A and turn only in one direction
- 6) Change the control coefficients ($K_{p,i,d}$) dynamically
- 7) Good resolution

2.2 PID and FPAA?

A couple of years ago FPGA made a tremendous break through in the digital field, in this project the analog counterpart was first meant to be used. Indeed, using programmable analog modules was advantageous for several aspects.

- 1) Everything could be done in the analog word and thus decrease the potential errors due to coding
- 2) FPAA allows an “on the fly” reconfiguration of the PID parameters
- 3) Challenge of using a new technology

Two companies (Anadigm and Lattice) are producing these chips, and claim that building PID modules is “easy”. However after having read several application notes and sent emails to design engineers who work for these companies the following was realized.

- 1) Implementing P, PI controllers is feasible easily, however PID requires external components, thus the whole point of using FPAA vanishes
- 2) Programming these chips uses SPI interfaces, which would not be troublesome if there was only one chip to configure. However, since there are 4 of them, translation, from RS232 to SPI, which should be avoided according to Anadigm's engineers, has to be performed

Knowing these two facts it was agreed that the FPAA technology was not yet mature enough to be used in a flying robot. We preferred to play it safe and build a module that would be a **fully analog PID** using standard OA **upgradeable to an FPAA PID**.

2.3 The chosen solution

The central PCB (figure 2.3a) does three main tasks, provide $\pm 12V$ that will be used for most IC (Integrated Circuits), generate an analog speed signal from a digital signal and generate a triangle wave signal for the PWM stage.

The following blocs can be seen on the central PCB:

- 1) An I2C input. The command signal is sent by the PC using the RS232 protocol which then is translated by a standard module to I2C and eventually by the DAC (Digital to Analog Converter) to an analog signal corresponding to the wanted speed. Note that in the future designs translation from RS to I2C will be performed on the central PCB, but could also be emulated by the central PC. The same connector also provides 5V and ground, which power the whole design.
- 2) A DAC, converts the I2C speed signal to an analog set point
- 3) A 5 to $\pm 12V$ converter: alimentation for most IC (Integrated Circuits)
- 4) A triangle wave generator, used for the PWM (Pulse Width Modulation)
- 5) Four Micromatch 6 connectors to the "PID modules", each containing $\pm 12V$, 5V, ground, triangle wave and command signal.
- 6) A voltage follower, to buffer the DACs output.

Note: on the figure below only the vital components are indicated.

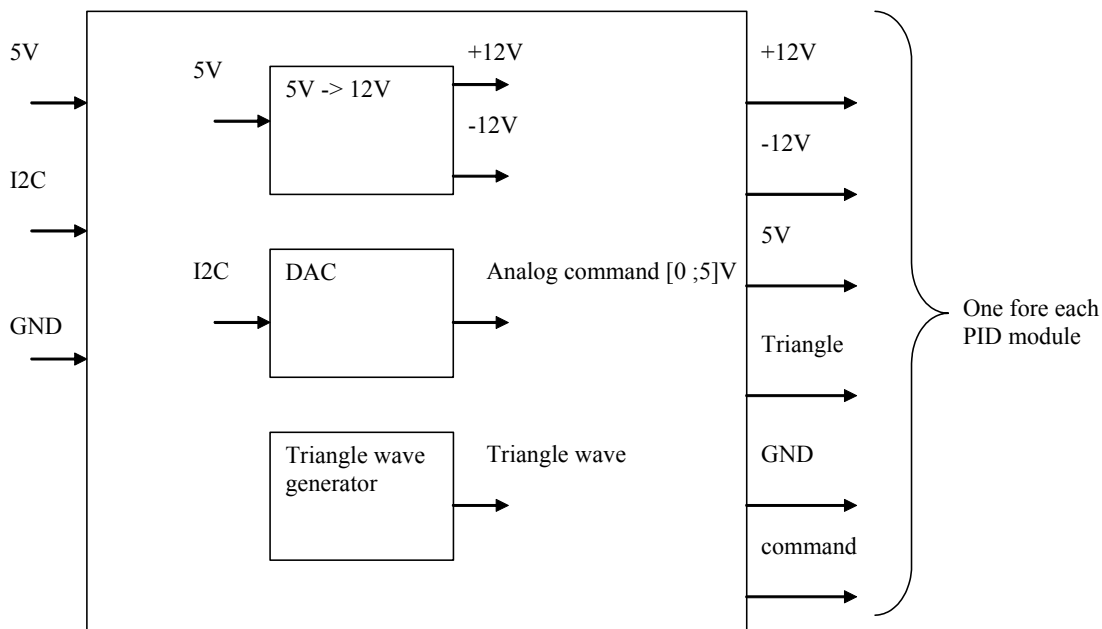


Figure 2.3a : central module

The PID PCB (figure 2.3b) handles the whole motor driving part as. Including the PID, PWM and so on.

On the PID module the following blocs can be observed:

- 1) A Micromatch 6 connector input from the central module
- 2) A connector for the battery
- 3) A connector to the motor
- 4) A differential amplifier, used to perform the “error = command – feedback”
- 5) An OA used as PID
- 6) An OA used to invert the PID’s output
- 7) Two comparators used to generate the PWM
- 8) A comparator to detect the sense of desired rotation
- 9) A full H-bridge, to drive the motor
- 10) A XOR gate to double the frequency read by the motor’s odometrical encoders
- 11) A frequency to voltage converter
- 12) A 2 pole Butterworth filter
- 13) Jumper allowing an open-loop operation
- 14) Jumper allowing to bypass the Butterworth filter
- 15) Jumper allowing to disable the reversion of the H-bridge (i.e. only one direction)

Note: on the figure below only the vital components are indicated.

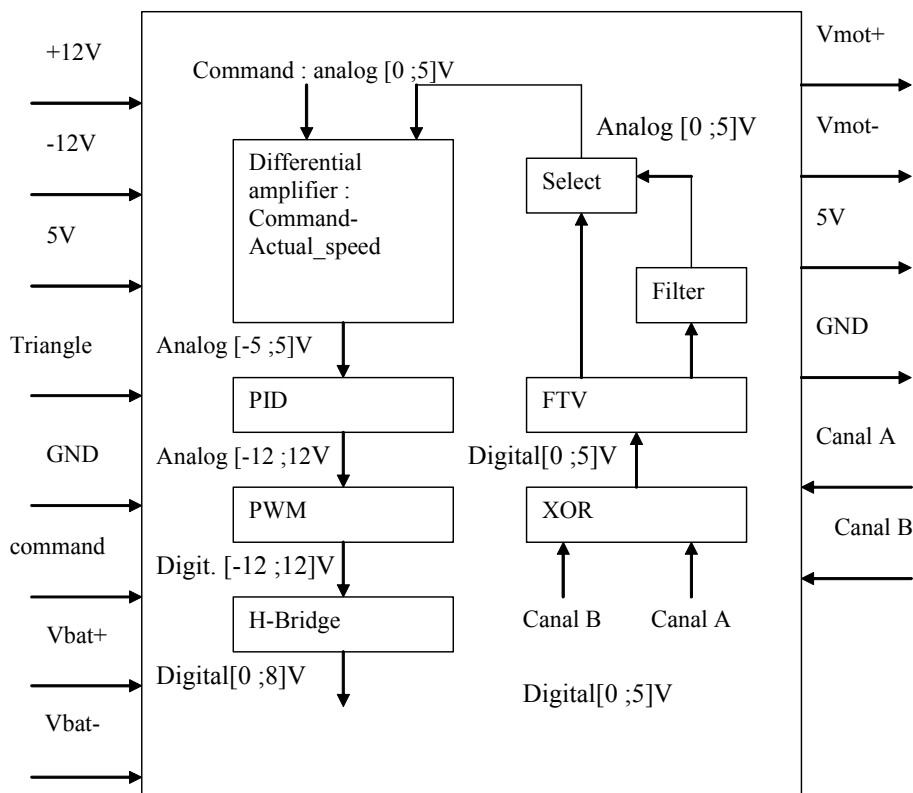


Figure 2.3b: PID module

2.4 Advantages of this solution

Even though this solution requires more chips than the current PIC using counterpart, most of these are available in SMD (Surface Mounted Device). For shipping reasons (or because they were requested as samples) some chips are unfortunately DIL packages (Dual In Line), but as stated before they EXIST in smaller packages.

The chosen design displays several attractive advantages. For instance theoretically one could fit the PID modules in the tubes of the OS4 flying robot and thus no space would be wasted and the drag properties of the robot would remain constant. This is only feasible if the routing is done properly and a two sided PCB might allow this kind of reduction, maybe several layers would be required.

Being fully analog implies a fast response of the PID, the current limiting part is the speed reading device.

Finally, no code is used, theoretically there is no possibility of crashing the helicopter because of a software bug. The down part of this is the impossibility to change $K_{p,i,d}$ on the fly. However jumpers allow to change these values within seconds.

2.5 Other possible solutions

Unfortunately FPAA was not used, however other techniques allow to change capacitor or resistor values more or less dynamically.

Programmable resistors could be used, however since a large number of these would be needed a PIC with an important number of digital outputs would be required to program all of them.

Several fixed value capacitors/resistors could be soldered and the signal routed using demultiplexers, again some PICs would be required and the achievable resolution would not be very high.

3 The central module: theory

In this section the central module will be explained in details.

3.1 5V -> $\pm 12V$

The “only” available voltage is 5V (from the I2C), which is insufficient for most applications, especially OA which require negative voltages as well. The MAX 761 was used to provide a stable (if the current used is constant, which is the case in our application) $\pm 12V$ source each at 100mA max.

Figure 3.1 shows how this chip was used in our application (source: Application Note from Maxim semiconductors).

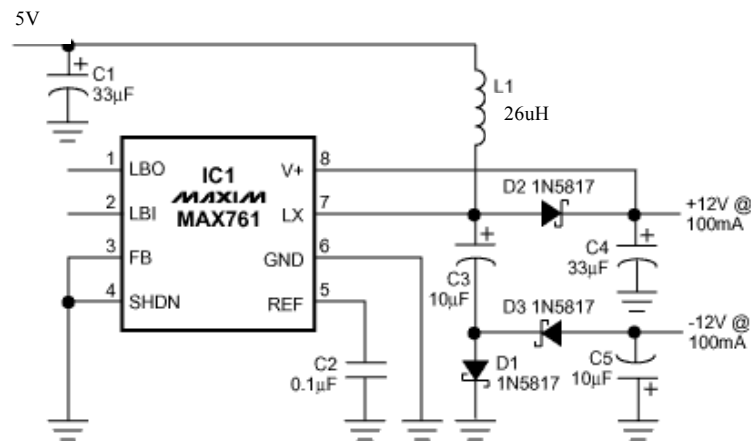


Figure 3.1 : MAX 761, 5V to $\pm 12V$

3.2 DAC

Setting the speed requires two translations. First from RS232 to I2C, which is done by the standard ASL translator. Second I2C to an analog signal, this step is done by the MAX 520. This chip has two key features:

- 1) 4 analog outputs
- 2) its I2C is compatible with the ASL I2C protocol

When using this chip the two pull-up resistors are used, the Maxim’s application note recommends using 1.7k Ω resistors, however to have a better signal values around 5k Ω are preferable.

Regarding the I2C communication. The first 7 bits encode the chips address, the 8 next the command (corresponds to the ASL register) and the last 8 bits the desired value. However the ASL I2C standard only sends 7 user programmable bits preceded by a 0 for the command signal, which fortunately corresponds to the 8 needed bits (the first bit MUST be a 0). This case is depicted on figure 3.2.

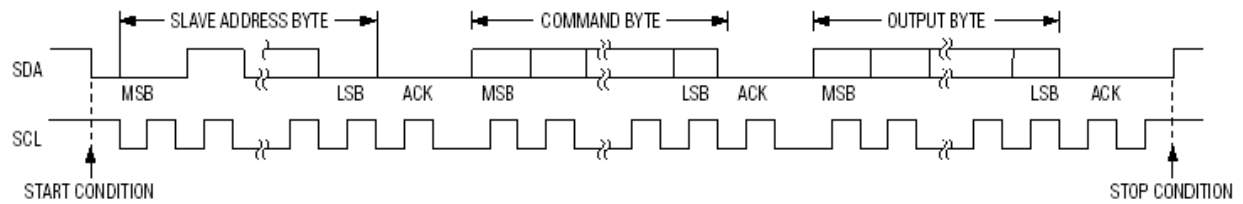


Figure 3.2 : MAX 520 and I2C communication

The **address** byte can be hard-coded by setting the value of 3 pins to high or low, in our case the resulting address is : 0x28 or bin 101000, where the 3 last bits are 0 because the corresponding pins are tied to the ground. The highest achievable address is 0x2F or bin 101111, with the last three bits high.

The **command** byte is simply the number of the desired output, 0x00 for output 0, 0x01 for output 1 and so on.

The **output byte** is the desired output value, 0x00 will yield a 0 V output and 0x255 to a 5V output, any dec value inbetween will result in a linear value between 0 and 5V.

$$V_{out} = 5 \cdot \frac{dec_value}{255}$$

Note: In order for this chip to work the output impedance has to be infinite (the chip can't deliver enough current). Thus a voltage follower is needed to buffer the output. Note that other chips are available.

3.3 Triangle wave

A triangle wave is required to generate the PWM signal to drive the H-bridge. This wave is simply generated by using one comparator and one OA. Figure 3.3 shows how to connect the different components.

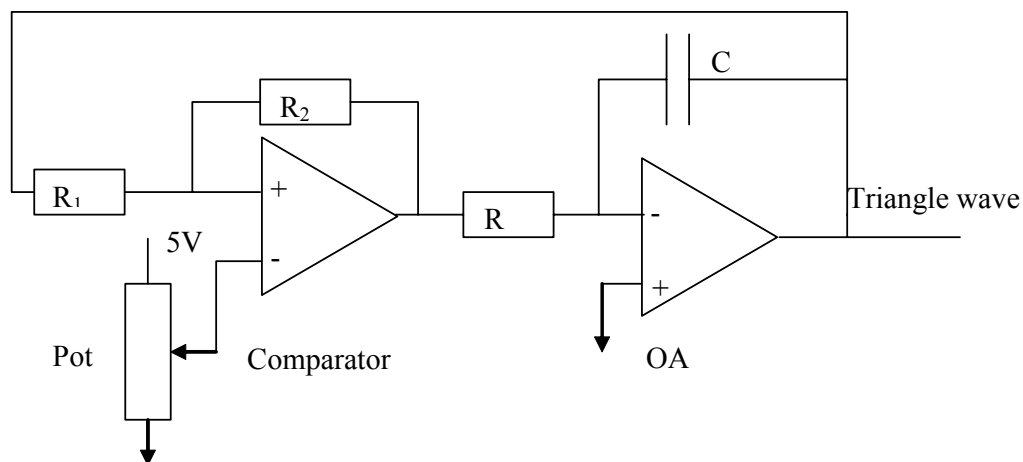


Figure 3.3a : triangle wave generator

Note: A pull-up resistor is required at the output of the first stage (comparator), for practical reasons it is not shown on figure 3.3a.

The amplitude of the triangle wave is given by:

$$\text{Amp} = 2 \cdot V_{\text{sat}} \cdot \frac{R_1}{R_2}$$

Since an *Amp* of 10V is wanted and V_{sat} is 12V the following values of R_1/R_2 have to be used:

$$R_1 = 22\text{k}\Omega \text{ and } R_2 = 52.8\text{k}\Omega.$$

In the practical case there is a potentiometer in series with R_1 in order to allow a fine tuning of the amplitude.

Dividing R_1 by two will result in a 5V amplitude.

The frequency is given by:

$$f = \frac{1}{T}$$

where:

$$T = 4 \cdot R \cdot C \cdot \frac{R_1}{R_2}$$

In order to have f greater than 20kHz the following values of R and C were used.

$$R = 47\text{k}\Omega \text{ and } C = 470\text{pF}.$$

Finally the potentiometer “Pot” is used to ensure that the triangle wave is between 0 and 10 V and not -5 to 5V. A reasonable value for this potentiometer is 5k.

The first PCB was designed in a way that allows an easy debugging; this is why several “jumpers” have been placed to allow an easy changing of, R (jumper 1) and C (jumper 2). There is also a possibility to change R_1 , despite the fact that it is not depicted on figure 3.3b.

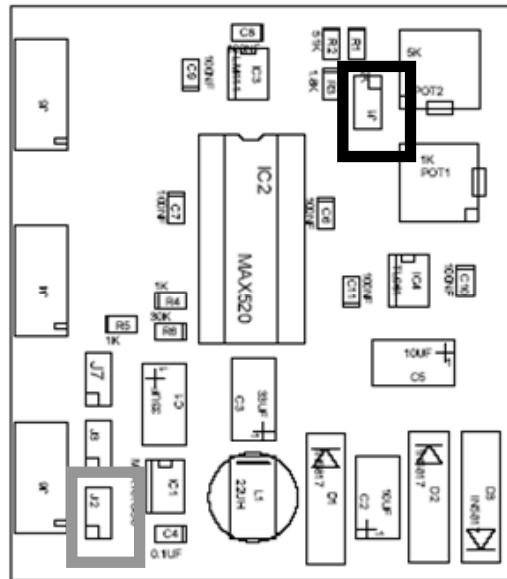


Figure 3.3b : PCB 1 jumpers (J1 = black)

3.4 Voltage follower

Figure 3.4 depicts a voltage follower. Notice that a simple OA is used.

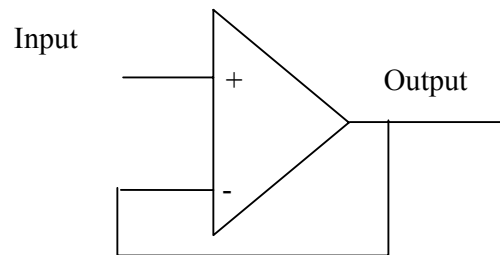


Figure 3.4 : voltage follower

As expected the transfer function is:

$$V_{out} = V_{in}$$

3.5 Summary of the used components

Component	Function	Used package	Existing package
Max 761	5V -> $\pm 12V$	SO 8	
Max 520	DAC	DIL 16 (sample)	SOIC 16
LM 111	Comparator	SO 8	SOIC
TL 061	OA triangle wave	SO 8	
LF 347	OA follower	DIL 14	

Note that most components exist in small to very small packages.

The MAX 520 is a DIL because it was requested as a sample.

The LF 347 is also a DIL because it was the only available OA in the lab.

4 The PID module: theory

This section will deal with the PID module, here one will find information about every building block on this PCB.

4.1 Error = command – feedback

Here an OA is used as a “differential amplifier”.

One might want to note that the OA used to do the PID is an inverter, for this reason the command signal is sent to the inverting input (V_1 on figure 4.1) and the error signal to the non inverting input (V_2 on figure 4.1) of the differential amplifier.

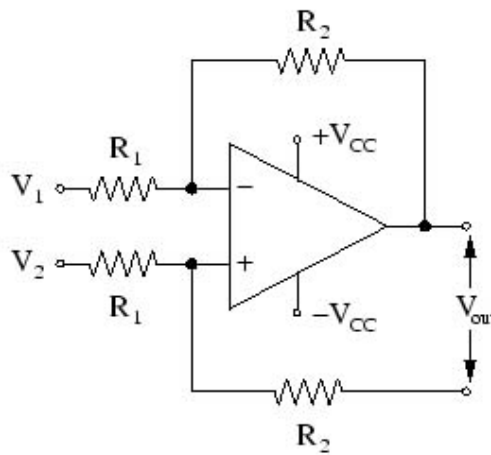


Figure 4.1 : differential OA

The values of $R_{1,2,3,4}$ have to be equal and are set to 10k Ω .

The resulting transfer function is:

$$V_{out} = V_2 - V_1$$

4.2 PID

Implementing a PID controller can be done by using a single OA. One knows that the transfer function of an OA is given by : $T = -\frac{Z_2}{Z_1}$, where Z_1 is the input impedance and Z_2 the feedback impedance as depicted on figure 4.2.

If one chooses Z_1 and Z_2 as depicted in figure 4.2a some simple math show that transfer function is:

$$\frac{V_{out}}{V_{in}} = - \left[\left(\frac{R_2}{R_1} + \frac{C_1}{C_2} \right) + R_2 \cdot C_1 \cdot S + \frac{1}{R_1 \cdot C_2} \cdot \frac{1}{S} \right]$$

Which corresponds to a PID controller.

Where:

$$K_p = \frac{R_2 + C_1}{R_1 \cdot C_2}$$

$$K_i = R_2 \cdot C_1$$

$$K_d = \frac{1}{R_1 \cdot C_2}$$

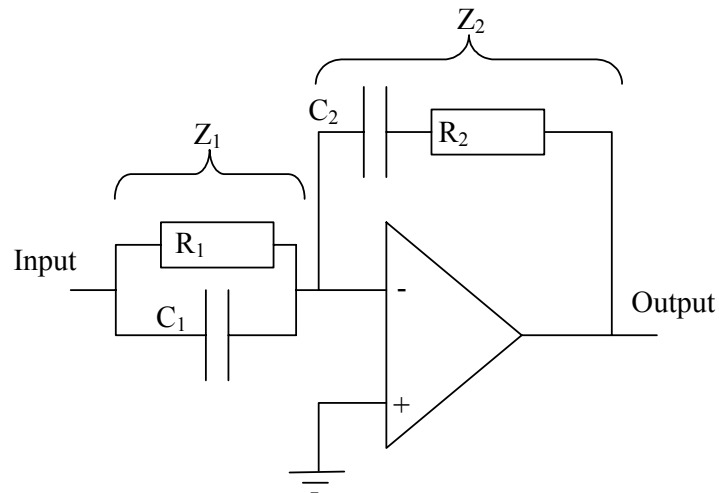


Figure 4.2a : PID controller

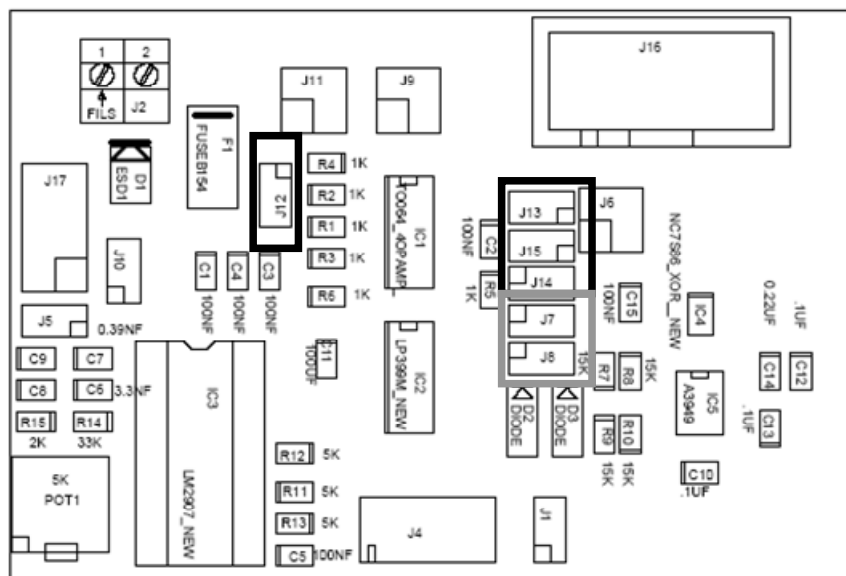


Figure 4.2b : PCB2 Kpid jumpers (black), acc. and dec. jumpers (gray)

For debugging reasons C_1 (jumper 12), C_2 (jumper 13), R_1 (jumper 14) and R_2 (jumper 15) have to be changed easily. This is done by using simple jumpers where one simply can plug in the select components. Refer to figure 4.2b.

4.3 Inverter

Inverting the PID's output is required to generate the correct PWM signal. This operation is performed by an OA connected as an inverter. As seen in section 4.2 the transfer function is given by $T = -\frac{Z_2}{Z_1}$, if one chooses $Z_2 = R_f = Z_1 = R_{in} = 10k\Omega$ the resulting configuration is an inverter as depicted on figure 4.3.

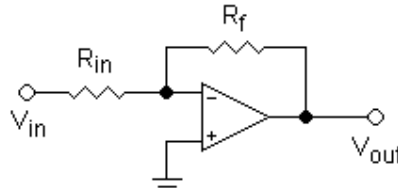


Figure 4.3 : inverting OA

The transfer function is simply:

$$V_{out} = -V_{in}$$

4.4 PWM

The motor's speed is controlled by PWM, explaining why this method has been chosen is beyond the scope of this report. One has to know that this corresponds to switching on and off the alimentation of the motor at a high-rate. In order to maximise efficiency and minimize noise this switching frequency has to be 20kHz or more.

In this case a PWM signal with the following characteristics was wanted:

- 1) pulse width has to increase (0% to 100%) as the output of the PID goes from 0 to 10V
=> **speed proportional to voltage**
- 2) pulse width has to increase (0% to 100%) as the output of the PID goes from 0 to -10V
=> **speed proportional to voltage**
- 3) have a separate line going high if the PID's output is high, and low otherwise

These three characteristics are obtained by using 3 comparators in the configuration depicted in figure 4.4a.

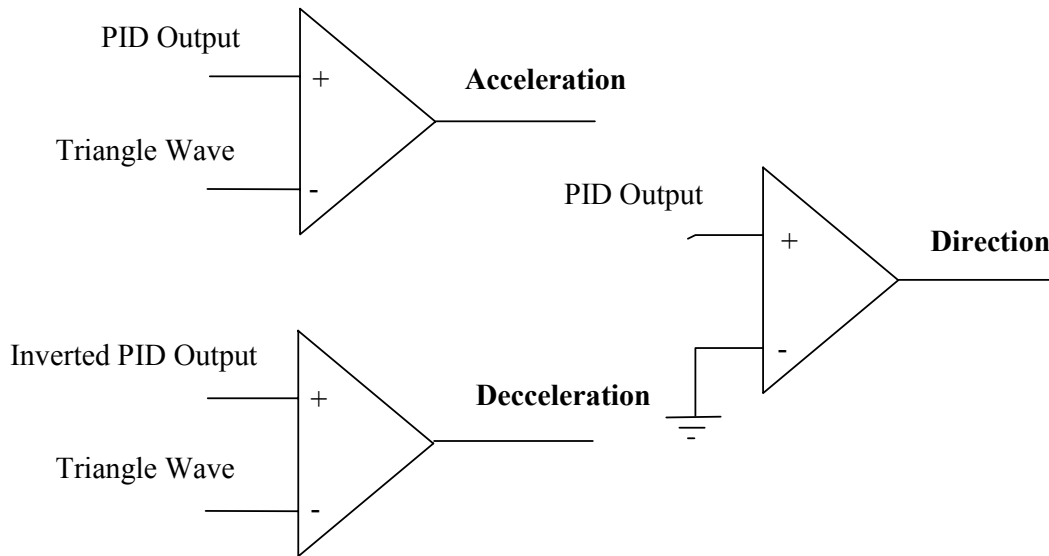


Figure 4.4a : generating PWM

Note: Pull-up resistors have to be placed at the output of each comparator. For practical reasons they are not drawn on figure 4.4a.

Before feeding these signals to the H-Bridge several steps have to be done.

First, *acceleration* and *deceleration* are summed, since only one will be high at a time this can be done by the configuration shown in figure 4.4b.

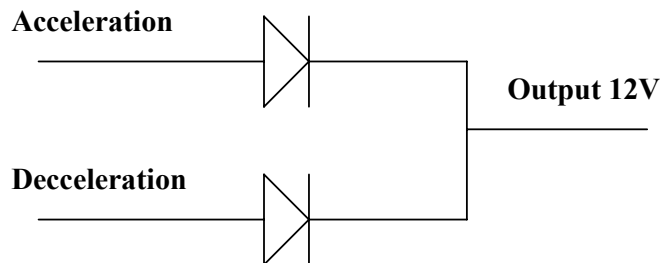


Figure 4.4b : summing acceleration and deceleration

Second, the H-Bridge specifications don't allow an input voltage of 12V, thus two voltage dividers have to be used to lower the signals to 7V or less.

Knowing that the maximal input current is 40uA and applying the rule of the thumb : "The current leaving the divider has to be equal to 1/10th of the current flowing to the ground" and Ohm's law one can easily find that the values of R_1 and R_2 on figure 4.4c have to be equal to 15k Ω . Actually since there is a 4.7k Ω at the output of the comparator (pull-up) $R_{1,2}$ will create a small voltage drop, thus the voltage in-between $R_{1,2}$ won't be 6V exactly but slightly lower.

Furthermore the input voltage of the H-bridge can't go below -0.7V, thus diodes had to be added as can be seen on figure 4.4c.

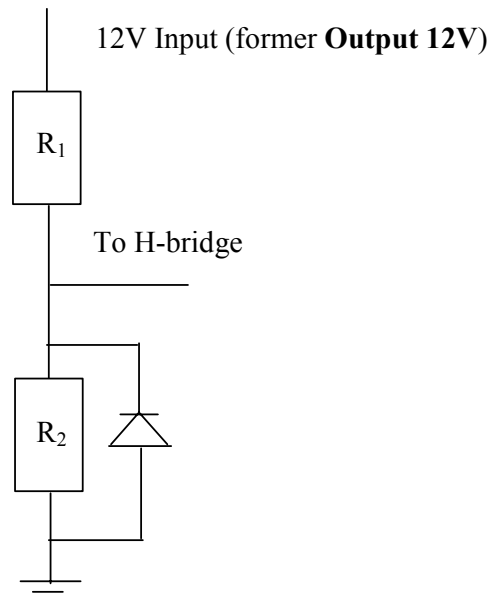


Figure 4.4c : voltage divider

Please note that on the practical version there are two jumpers allowing to enable or disable the passage of the “acceleration” and “deceleration” signal (jumper 7 and 8 on figure 4.2b). This feature is especially useful for debugging.

4.5 H-Bridge

H-bridges are devices that are usually used to drive motors. They come in different configurations, half-bridges, full-bridges, They allow to drive a motor in both directions using a single power supply and this with a high efficiency.

The one chosen for this specific application had to allow high current and low voltages, both these criterias are fulfilled by the A 3949 chip.

Having a look at figure 4.5 shows that there are 4 logic inputs. After examination of the Control Logic Table it is found that:

MODE: has to be set to high, a slow decay braking is wanted
PHASE: has to be connected to the output of the direction detector
ENABLE: has to be connected to the PWM signal
SLEEP: has to be set high

Note: The operation mode obtained above is not wanted with the given desing. As will be seen later due to physical limitations there is an offset in the frequency to voltage conversion. Resulting in perturbations that cause the motor to “go crazy”.

To overcome this the following logic levels are wanted. On the demo PCB however one can easily switch from one to the other.

Observe that only the acceleration signal is fed to the H-bridge. This is logical, because phase is always high the H-bridge can physically not “reverse” the voltage at the output of the motor. An thus the deceleration PWM would be handled as an acceleration PWM.

MODE: has to be set to high, a slow decay braking is wanted
PHASE: is set high, turns only in one direction
ENABLE: has to be connected to the PWM signal, but only the accelerating signal
SLEEP: has to be set high

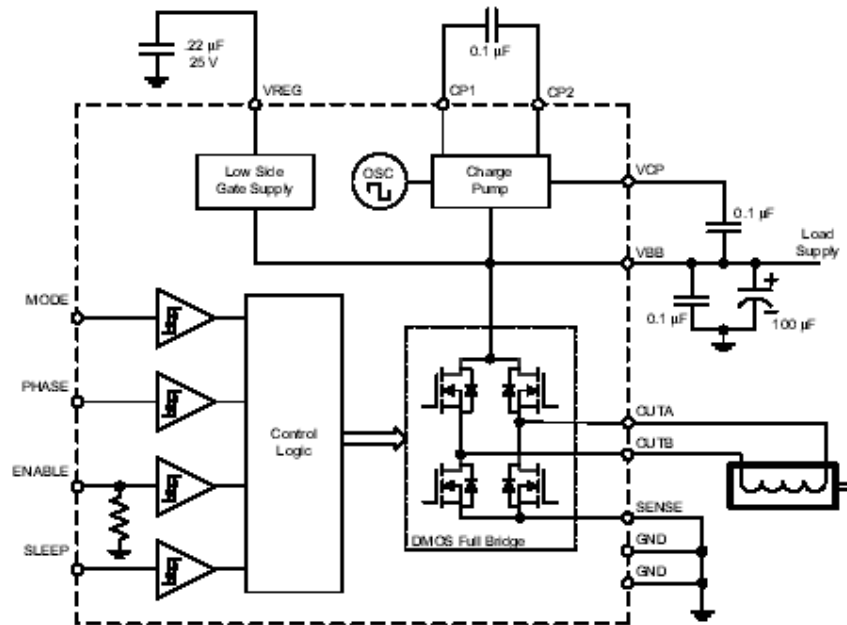


Figure 4.5 : H-bridge

4.6 Motor

For testing purposes we used a 1724006 SR from Faulhaber with a 3.7 reduction. This motor is slightly less powerful than the one used on the real flying robot.

4.7 Speed encoder

Explaining in details how a speed encoder works is not the purpose of this report. Simply keep in mind that they usually use optical encoders to sense the speed. Generally two outputs are provided, canal A and canal B, both come from detectors shifted by 90°, and thus allow to detect not only the **speed** of the motor but also by using the proper decoding hardware the **direction of rotation**.

For practical reasons a Ie2-512 from Faulhaber was used, which provides 512 increments per rotation, whereas the encoder used on the OS4 only has 16. This is why calculations were done for both.

4.8 Frequency doubling

When performing frequency to voltage conversion (FTV) one is facing a hard fact: “faster response time, more noise at the output”. It is known that in these configurations noise is dependent of the input frequency, and one can take advantage of this. Indeed as the frequency goes up, noise goes down. Thus doing an XOR on canal A and canal B (see section 4.7) can double the frequency and thus reduce the noise.

This feature was bypassed during the test. As stated in section 4.7 the used motor had too many increments, thus a doubling was highly unwanted.

4.9 Frequency to voltage

Frequency to voltage conversion is usually performed by a PIC microprocessor. Either by measuring the time between two falling edges or counting the number of edges in a given time period.

In this specific design a fully analog system is wanted. Several chips are available for doing this operation, however some are big, some require the input voltage to swing below 0V to trigger the detection and most are not available easily.

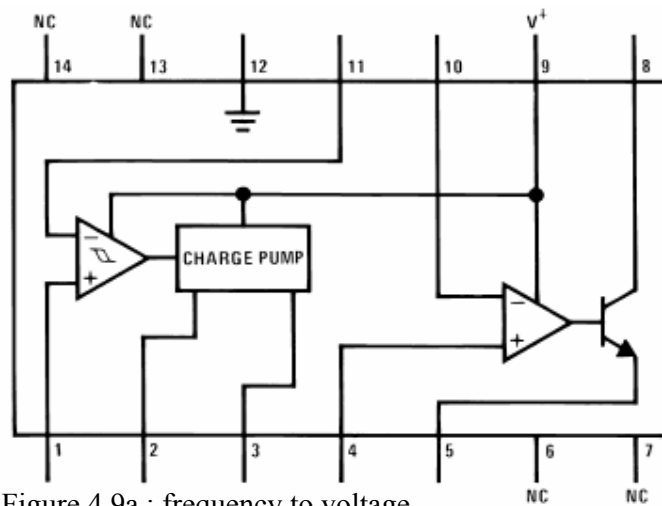


Figure 4.9a : frequency to voltage

Eventually the LM 2907 was chosen. Again here one will only find information about how to get the chip to work. Basically every time the input signal swings below a reference voltage (PIN 11 on figure 4.9a) the charge pump gets active and the voltage read on PIN 3 changes. The following 4 PINs are important for this operation.

PIN 1: input PIN 2: C_1 to ground PIN 3: C_2 in parallel with R_1 to ground (filter) PIN 11: at 2.5 V
--

There is an OA on the LM 2907, which will be used to buffer the output, otherwise the output impedance would be C_2 in parallel with R_1 . To do so one simply needs to tie PIN 3 to PIN 4 and PIN 5 to PIN 10. Also connecting PIN 8 to V_{cc} and tying a resistor of $5k\Omega$ between PIN 5 and the ground is required. This is illustrated in figure 4.9b.

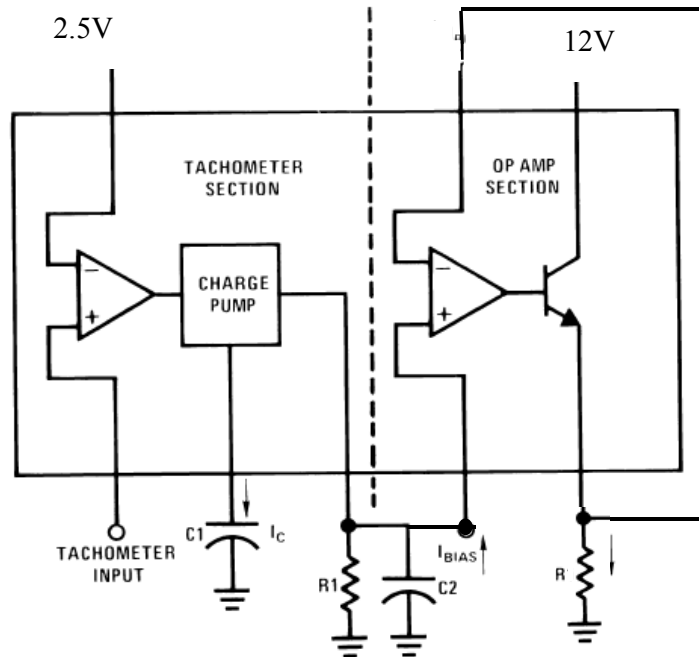


Figure 4.9 b : final configuration

Several considerations have to be taken when choosing the values of C_1 , C_2 and R_1 . There is a trade off to be made between noise and response time. Since we have a 8bit DAC on 5V, the resolution will be 20mV. Noise should be kept below half of this value. And calculations in “André Noth’s report” have shown that the response time of the motor/propeller system was around 300 ms. We want our conversion to be done in 1/10th of this, 30ms. Furthermore the maximal frequency will be 3200Hz.

The output voltage is given by :

$$V_{out} = V_{sup} \cdot R_1 \cdot C_1 \cdot f_{in}$$

where $V_{sup} = 12V$.

With these constraints and some math the following values: $R_1 = 27k\Omega$, $C_1 = 3.69nF$ and $C_2 = 0.84\mu F$ are found.

However as stated before this is for the “real” motor, the one used for debugging has a maximal frequency around 32kHz (corresponding to a DAC value of 220), under these conditions $R_1 = 28k\Omega$, $C_1 = 388pF$. Unfortunately C_1 should be higher than 500pF, but with the given constraints there is no way to bypass this value. This results in some nonlinearity in the conversion as well as a small offset if $f_{in} = 0$, which should disappear when the right motor will be used.

Note 1: 30 ms might not be fast enough, so jumpers are available to bypass the filtering capacitor C_2 and use an alternative 2nd order Butterworth filter. In order to use this filter on needs to **unplug J5** which is connected to C_2 and plug J11 as depicted on figure 4.9c. For using the filter do the opposite.

Note 2: a 20kΩ pot is in series with R_1 in order to allow a broad tuning.

Note 3: physical constraints are limiting the maximal admissible speed of the motor. However once the real motor will be used the DAC will be able to go up to 255 and the maximal velocity will be higher

Note 4: the noise is given by the following equation:

$$V_{noise} = \frac{V_{sup}}{2} \cdot \frac{C1}{C2} \cdot \left(1 - \frac{V_{sup} \cdot f_{in} \cdot C_1}{I_2} \right)$$

where $I_2 = 180\mu A$

Note 5: there is also a limitation on R_1 given by:

$$R_1 \geq \frac{V_{out}}{I_2}$$

Where V_{out} is the maximal output voltage of the FTV, 5V in our case.

Note 5: the maximal achievable input frequency is given by:

$$f_{max} = \frac{I_2}{C_1 \cdot V_{sup}}$$

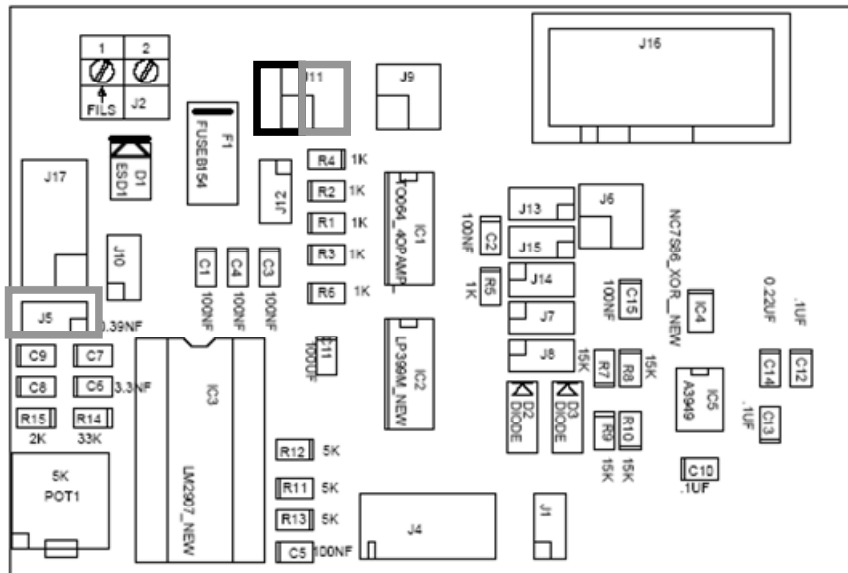


Figure 4.9c : jumper 5 and jumper 11 (black = butterworth enabled)

4.10 Butterworth filter

As stated in section 4.9 improving the response time of the frequency to voltage conversion can be done by using a Butterworth filter instead of the on-chip C_2 capacitor.

Doing this allows to have the same amount of ripple but a setting time quite lower.

Implementing a second order filter can be done by different means. One is to use an OA with the Sallen-Key topology as depicted on figure 4.10.

Analog PID module

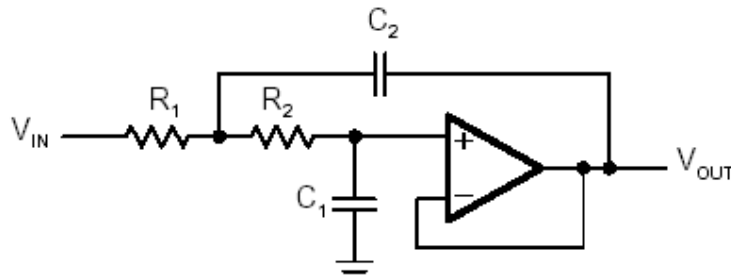


Figure 4.10 : Sallen-Key unity gain

The design procedure from : www.elkraft.ntnu.no/~sie10ah/ActiveFilterDesign.pdf (page 15) was used.

The values of $R_{1/2}$ and $C_{1/2}$ are determined by the following equations:

$$R_{1,2} = \frac{a_1 C_2 \pm \sqrt{a_1^2 C_2^2 - 4b_1 C_1 C_2}}{4\pi f_c C_1 C_2}$$

$$C_2 \geq \frac{C_1 4b_1}{a_1^2}$$

The values of a_1 and a_2 are given by the order of the filter, one finds: $a_1 = 1.41$ and $a_2 = 1$.

A f_c of 100Hz yields the following values:

$$\begin{aligned} R_1 &= 10\text{k}\Omega \\ R_2 &= 2.7\text{k}\Omega \\ C_1 &= 100\text{nF} \\ C_2 &= 200\text{nF} \end{aligned}$$

For debugging reasons one might want to be able to change C_1 , C_2 , R_1 and R_2 easily. This is done by using some jumpers which allow to plug the select component easily. The location can be found on figure 4.11 (J17 in dashed).

Note: Recall that to use this filter one needs to set the jumpers correctly (report to figure 4.9c).

4.11 Open loop control

One might want to use open loop control, this is feasible by setting the jumpers correctly. Basically one directly feeds the “command signal” to the input of the PWM generator. In order to use this feature one has to set the triangle wave amplitude to 5V by changing the value of R_1 (refer to section 3.3) and set the jumper 9 as shown on figure 4.11.

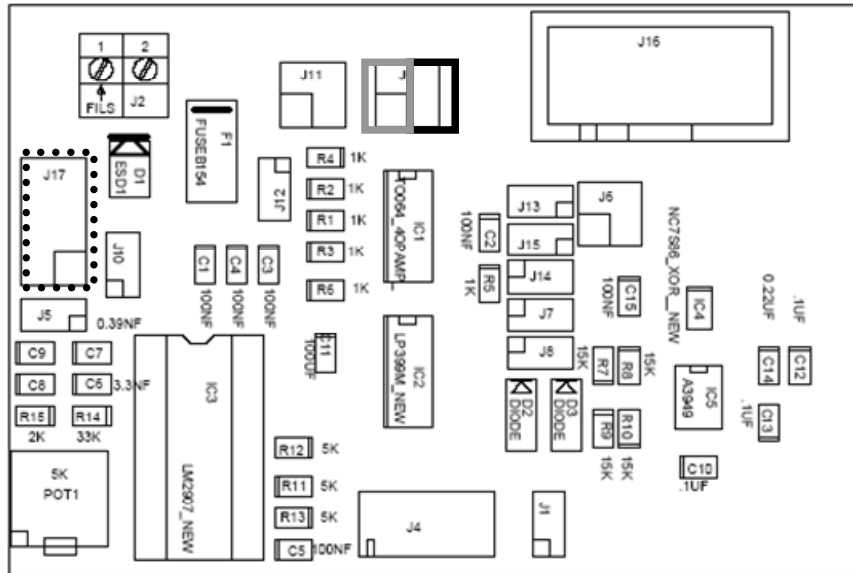


Figure 4.11 : open-loop control (black = open loop, gray = feedback)

4.12 Fuse

In order not to destroy the motor or H-bridge a fuse was installed as well as a diode that would trigger the fuse burn if the voltage was reversed (bad manipulation) or the current too high. Figure 4.12 depicts these two features. Of course the fuse is changeable easily.

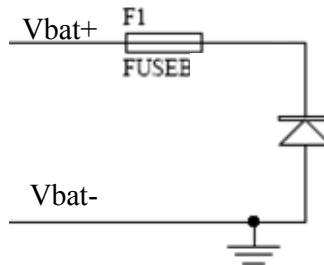


Figure 4.12 : fuse and diode

4.13 Summary of the used components

Component	Function	Used package	Existing package
TO 064	Quad OA	SO 14	
LM 399M	Quad Comparator	SO 14	
A3949	H-bridge	TSSOP 16	
NC7S86	XOR Gate	SOT 23	
LM 2907	FTV Converter	DIL 14	SO 14
LF 347	Quad OA	DIL 14	

Note that most components exist in small packages.

Analog PID module

The TO 064 was eventually replaced by a LF 347 which was the only available OA in the lab, thus the DIL package.

The FTV converter is a DIL, because standard distributors are not selling the smaller package.

5 Designing the controller ($K_{p,i,d}$)

A long time ago, people had to design all the controllers by hand, which required drastic simplifications and a great knowledge. Several techniques exist in the continuous time domain such as, root locus and frequency response. Nowadays one usually models the system and “fiddles the numbers” using Matlab.

Due to time constraints only the second method will be fulfilled.

5.1 Simulation model for matlab

The following equations are borrowed from Andre Noth’s diploma project.

One can find the angular acceleration of motor, reductor and load in rad/s using the following equation.

$$\dot{\omega}_m = -\frac{1}{\tau} \omega_m - \frac{d}{\eta r^3 J_{tot}} \omega_m^2 + \frac{1}{k_m \tau} u$$

where:

$$\frac{1}{\tau} = \frac{k_m^2}{R J_{tot}}$$

And:

$$J_{tot} = \frac{J_{hélice}}{\eta r^2} + J_m$$

with:

$J_{hélice} = 5.1 \cdot 10^{-5}$ propeller’s inertia

$\eta = 0.9$ reductor’s efficiency

$r = 3.7$ reduction ratio

$J_m = 1 \cdot 10^{-7}$ motor’s inertia

$K_m = 3.59 \cdot 10^{-3}$ torque constant

$R = 3.41$ motor’s terminal resistance

$J_d = 9 \cdot 10^{-7}$ propeller’s drag factor

Thus the first equation becomes:

$$\dot{\omega}_m = -0.899 \cdot \omega_m - 0.00477 \cdot \omega_m^2 + 250 \cdot u$$

Note: the inertia of the gearbox was not modelled, since it was supposed to be small in comparison to the propeller’s inertia, even seen through the reductor stage.

5.2 Creating the simulink function

In an m.file the three following lines are needed.

```
omega=in(2); // actual speed (from the output of the box)
u=in(1); // desired speed in V
out=(-0.89*omega-0.00477*omega^2+250*u); // output acceleration in rad/sec2
```

5.3 Interfacing with Simulink

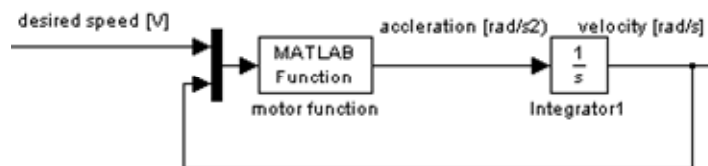


Figure 5.3: simple simulink model

The desired speed is in volt and is comprised in the $[-V_{\text{bat}}, V_{\text{bat}}]$ interval.
 The function's output is equal to the motor's angular acceleration in rad/s^2 .
 The integrator's output is the motor's angular speed in rad/s .

This “bloc” can easily be interfaced with a closed loop system as one will find out in section 5.4.

5.4 Complete simulink model

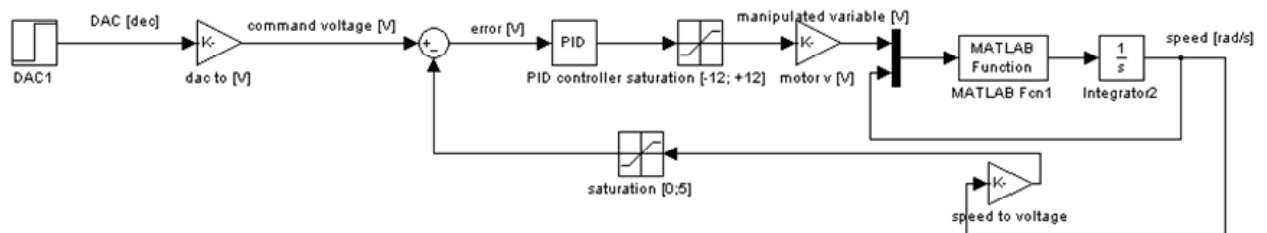


Figure 5.4: simple simulink model

In order to make the simulation more realistic note that there is saturation in the feedback loop, indeed the FTV converter can't go above a certain voltage.
 And since the alimentation voltage of the motor is not equal to the PID's output a gain stage has to be inserted (maximal PID output = 12V for a maximal $V_{\text{bat}} = 8\text{V}$ for this motor).

5.5 Steady state transfer function (open loop)

One might want to know the motors steady state speed knowing the DAC input. One way to do this is to run the simulation in simulink, in this case refer to section 5.4. Or to calculate it. Please remark that this way of calculating is only true for the open-loop case, the close loop case will be handled later.

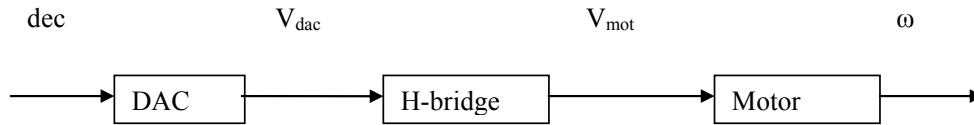


Figure 5.5a open-loop final velocity

The whole system can be broken down to the three parts depicted on figure 5.5a. The input is a decimal number that will go out of the DAC as a voltage that will then drive the H-Bridge, which will output a proportional voltage and eventually the motor that will output an angular velocity.

Where:

$$V_{dac} = V_{ref} \cdot \frac{dec_value}{255}$$

$$V_{mot} = V_{dac} \cdot \frac{V_{sup\ ply}}{V_{ref}}$$

$$\dot{\omega}_m = -\frac{1}{\tau} \omega_m - \frac{d}{\eta r^3 J_{tot}} \omega_m^2 + \frac{1}{k_m \tau} V_{mot}$$

Combining the two first and setting the angular acceleration in the third equal to zero yields and solving for the angular speed yields:

$$\omega_{mf} = \frac{-\frac{1}{\tau} + \sqrt{\frac{1}{\tau^2} + 4 \frac{d \cdot dec_value \cdot V_{sup\ ply}}{255 \eta r^3 J_{tot} k_m \tau}}}{2 \left(\frac{d}{\eta r^3 J_{tot}} \right)}$$

Where ω_{mf} is the steady state motor velocity in rad/s for the corresponding decimal input.

In order to test if this equation was right it was compared with the results obtained by the simulink's simulation. Figure 5.5b shows a perfect match of the two curves, actually only one curve is observable.

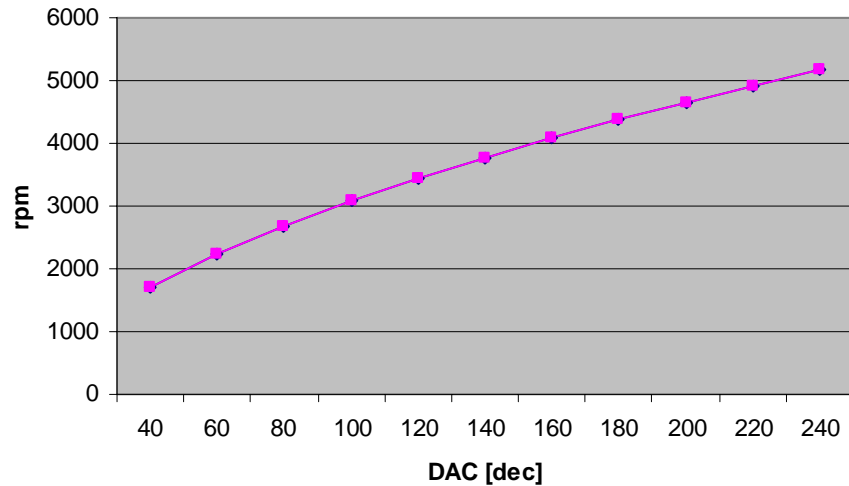


Figure 5.5b: equation and simulink

5.6 Steady state transfer function (closed loop)

To find the steady state speed of the motor in closed loop given a certain DAC input one simply sets the command voltage equal to the FTV voltage and solves for the frequency. Below one will find the key equations as well as the final equation.

$$V_{dac} = V_{ref} \cdot \frac{dec_value}{255}$$

$$V_{out} = V_{sup} \cdot R_1 \cdot C_1 \cdot f_{in}$$

$$RPM_{mot} = f_{in} \cdot \frac{60}{N}$$

Where N is the number of pulses per rotations in the speed encoder.

Eventually:

$$RPM_{motor} = \frac{dec_value \cdot V_{ref} \cdot 60}{255 \cdot R_1 \cdot C_1 \cdot N \cdot V_{sup}}$$

5.7 Conversion table

5.7.1 Closed-loop

The following table gives the conversion from the DAC input to the steady state output in different units.

DAC	V _{dac}	V _{ftv}	Canal A	rpm (mot.)	rpm (prop.)	rad/s
0	0.0	0.0	0.0	0.0	0.0	0.0
20	0.4	0.4	3103.0	363.6	98.3	38.1
40	0.8	0.8	6206.1	727.3	196.6	76.2
60	1.2	1.2	9309.1	1090.9	294.8	114.4
80	1.6	1.6	12412.1	1454.5	393.1	152.5
100	2.0	2.0	15515.2	1818.2	491.4	190.6
120	2.4	2.4	18618.2	2181.8	589.7	228.7
140	2.7	2.7	21721.2	2545.5	688.0	266.8
160	3.1	3.1	24824.2	2909.1	786.2	304.9
180	3.5	3.5	27927.3	3272.7	884.5	343.1
200	3.9	3.9	31030.3	3636.4	982.8	381.2
220	4.3	4.3	34133.3	4000.0	1081.1	419.3
240	4.7	4.7	37236.4	4363.6	1179.4	457.4
255	5.0	5.0	39563.6	4636.4	1253.1	486.0

Where:

$$V_{dac} = V_{ref} \cdot \frac{dec_value}{255}$$

$$V_{FTV} = V_{dac}$$

$$rad/s = \text{simulink or equations from section 5.7}$$

$$rpm = \frac{60 \cdot [rad / s]}{2 \cdot \pi}$$

$$Canal_A = \frac{512 \cdot [rpm]}{60}$$

5.7.2 Open-loop

The following table gives the expected output for a given DAC input.

DAC	Vdac	Vftv	Canal A	rpm (mot.)	rpm (prop.)	rad/s
0	0.0	0.0	0.0	0.0	0.0	0.0
20	0.4	0.4	8978.9	1052.2	526.1	110.1
40	0.8	0.8	14592.7	1710.1	855.0	179.0
60	1.2	1.2	19052.6	2232.7	1116.4	233.7
80	1.6	1.6	22868.3	2679.9	1339.9	280.5
100	2.0	2.0	26257.6	3077.1	1538.5	322.1
120	2.4	2.4	29338.0	3438.0	1719.0	359.8
140	2.7	2.7	32181.0	3771.2	1885.6	394.7
160	3.1	3.1	34834.4	4082.2	2041.1	427.3
180	3.5	3.5	37331.6	4374.8	2187.4	457.9
200	3.9	3.9	39697.4	4652.0	2326.0	486.9
220	4.3	4.3	41950.5	4916.1	2458.0	514.5
240	4.7	4.7	44105.7	5168.6	2584.3	541.0

rad/s was calculated by using equation:

$$\omega_{mf} = \frac{-\frac{1}{\tau} + \sqrt{\frac{1}{\tau^2} + 4 \frac{d \cdot dec_value \cdot V_{sup\ ply}}{255 \eta r^3 J_{tot} k_m \tau}}}{2 \left(\frac{d}{\eta r^3 J_{tot}} \right)}$$

For the conversions the same rules as in section 5.7.1 apply.

6 Central module: tests

In the following sections results about the different parts that compose the central PCB will be provided.

6.1 5 -> $\pm 12V$

The voltage conversion performs as expected.

Using a 120Ω resistor allowed to prove that it was possible to provide a 12V (100mA) alimentation without a noticeable voltage drop.

The same test was done for the -12V side.

Using two 270Ω resistors proved that $\pm 12V$ could provide each around 50mA at the same time.

In these tests the consumption of PCB1 was not taken into account, however knowing that it used $\pm 12V$ only proves that the DC-DC converter can even provide more.

6.2 DAC

Testing the DAC was done in two steps because, the ASL I2C protocol is quite constraining and one had to make sure that it was compatible with the given DAC. Communication with the DAC is performed with PIC Watch.

6.2.1 Using a Hirschman board

The first tests were done on a Hirschman board using the MAX 519 (same as the MAX 520 but with two outputs instead of four).

As expected the tests were concluding and thus the same implementation was chosen to be done on the final PCB.

Figure 6.2.2 shows as expected a 5V output for a DAC input of 255.

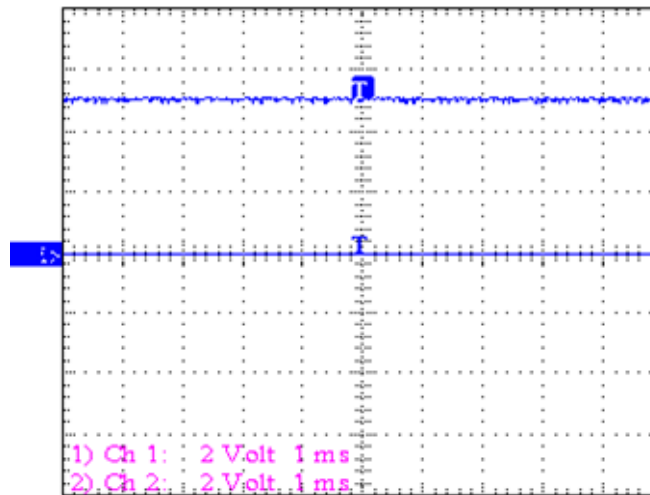


Figure 6.2.1 : max DAC output

6.2.2 On the PCB

To communicate with the MAX 520 one simply has to enter the right address and the correct commands (refer to section 3.2) and set the baud rate to 19200 in PIC WATCH.

Tests were conducted and figure 6.2.2 shows a quite honourable linearity.

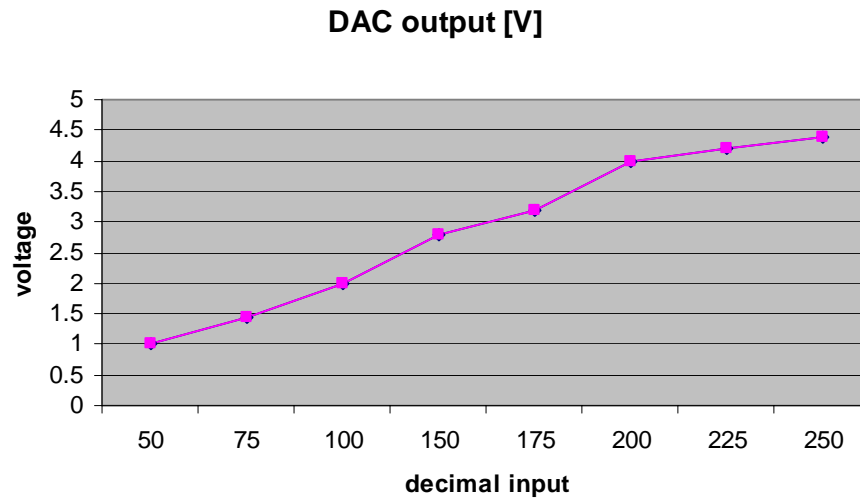


Figure 6.2.2 : DAC output

Unfortunately this chip is not able to provide any current (in contrary to MAX 519), this is why a voltage follower (refer to section 3.4) had to be bonded to the output.

6.3 Triangle wave

On figure 6.3a one can see the generated triangle wave with R_1 set to $20k\Omega$, as expected the amplitude is 10 Volts and the frequency is around 28kHz.

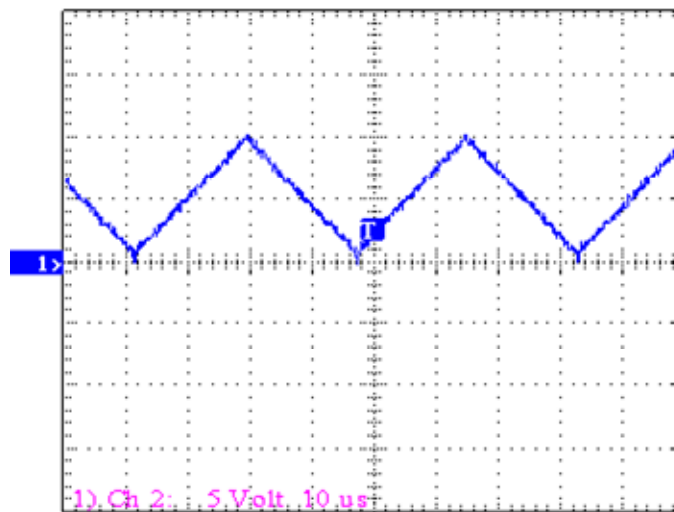


Figure 6.3a : triangle wave at 10V

On figure 6.3b one can see the generated triangle wave with R_1 set to $10k\Omega$, as expected the amplitude is 5 Volts and the frequency is around 57kHz.

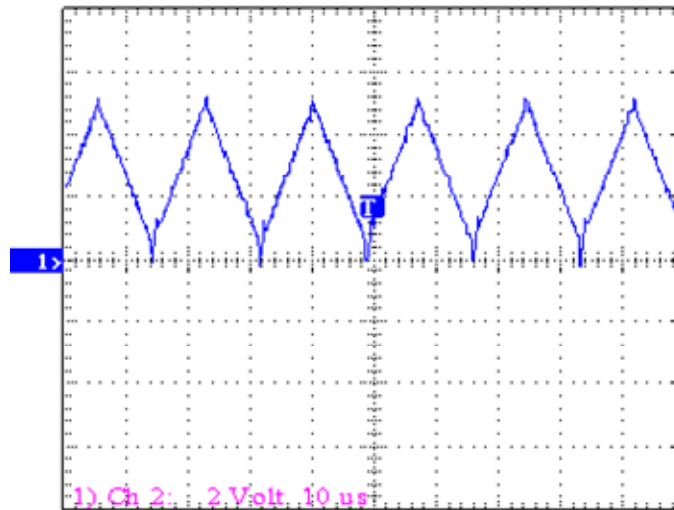


Figure 6.3b : triangle wave at 5V

As stated in section 3.3 amplitude, offset and frequency can be tuned easily.

6.4 Consumption

Item	Consumation	Comment
Generating triangle	60mA @ 5V	Uses 12V => conversion
DAC	Unnoticeable	Uses 5V
Voltage follower	30mA	Uses 12V => conversion

Some comments have to be made about the table above.

First, it is impossible to disconnect everything, so basically when only the triangle generator is left and of course the 5 to 12V converter. Then the whole PCB uses 60mA at 5V.

Second, the DAC is supposed to consume almost nothing since he is supposed to see a high impedance at its outputs.

Third, the voltage follower is consuming 30mA at 5V (but is alimented in 12V), this value seems excessive. The “chosen” component should be changed.

7 PID module : tests

In the following sections results about the different parts that compose the PID PCB will be provided.

7.1 Error = command - feedback

As explained in 6.1 the error in this case is given by : $\text{error} = \text{feedback} - \text{command}$ (recall that the PID is an inverter, thus the error has to be inverted).

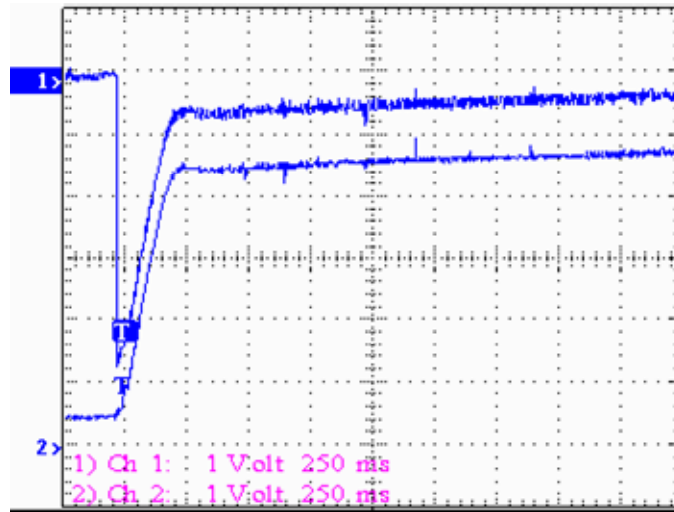


Figure 7.1 : error = feedback - command

Figure 7.1 shows two curves. The upper one is the error signal and the lower is the feedback (the command is set equal to 5V).

As predicted when the 5V command (step) is applied “error” sinks to -5V and as feedback grows error goes back to zero.

7.2 PID

Chapter 10 focuses on the PID controller.

7.3 PWM

In order to illustrate that the PWM is functioning the DAC was set to 126 out of 255, which should yield a 50% PWM (accelerating signal, these tests were done in “open loop”).

The result can be observed on figure 7.3.

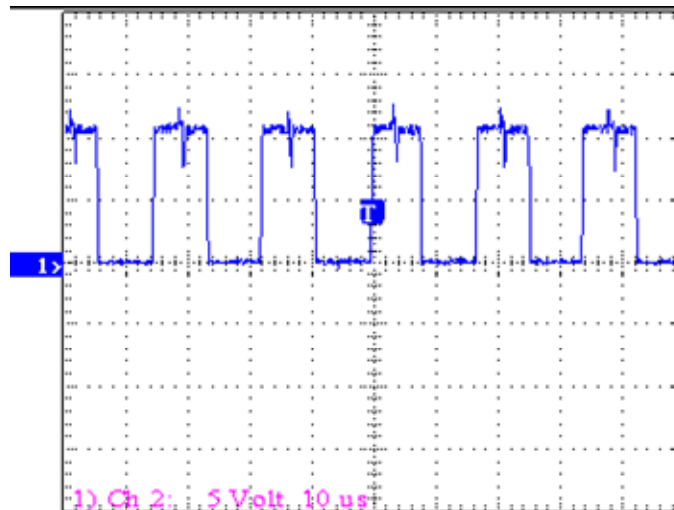


Figure 7.3 : 50% PWM

As expected the corresponding “decelerating signal” was equal to 0.

7.4 H-bridge

The H-bridge is performing as expected in the sense that the motor is accelerating and decelerating as requested.

It can deliver 400mA at 6.2 V without any problem.

Using a bigger motor allowed to show that delivering up to 1A at 18V was also possible.

Resistance tests (i.e. destruction of the H-bridge) could have been performed, but due to the limited amount of components available (samples) these were not conducted.

Note that the available motor was limited to 6V and the minimal supply voltage for the H-bridge is 8V. On figure 7.4 one can see the PWM output of the H-bridge (50%) if the alimantation is 6.2V.

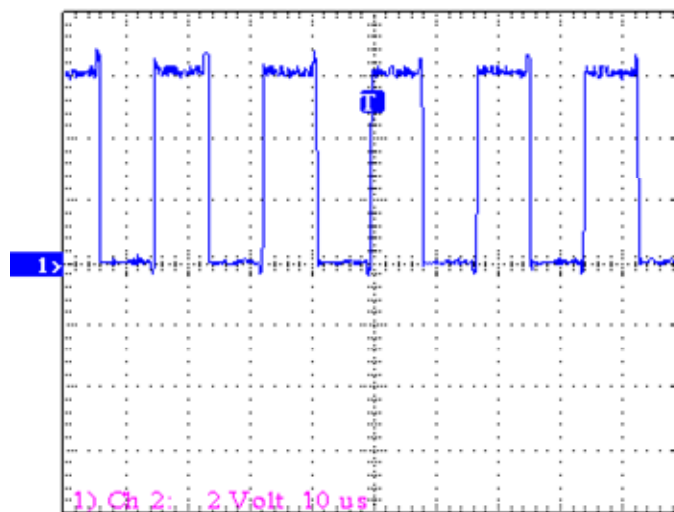


Figure 7.4: 50% PWM out of H-bridge

Tests have shown that to have enough current (i.e. for the H-bridge to work properly) the alimantation voltage had to be above 8V. Since we avoided to drive the motor too badly this was not a problem.

7.5 Motor

The given motor was a Faulhaber 1724006 SR coupled with a 3.7 reductor.

In order to make the tests more realistic a propeller was fixed to the extremity.

Tests were performed and it was shown that at full speed (i.e. DAC set to 255 out of 255) at $V_{\text{bat}} = 6.2 \text{ V}$ it would consume 0.35A and one channel of the speed encoder would display a frequency of 28kHz, which corresponds to about 3350 rpm.

If the motor is pushed to its limits (i.e. 0.7 A) it would spin at about 8000rpm. This shows that the read value of 3350 rpm for 0.35A is in the right range.

Later tests were performed with $V_{\text{bat}} = 8 \text{ V}$ (minimal value for the H-bridge), note that under these conditions the motor can go up to 41kHz (i.e. 4882rpm if DAC set to 255). However the speed had to be limited 32kHz (i.e. DAC set to 220), to minimize the current flowing through the coils (0.850A is too high) and obtain a proper FTV conversion.

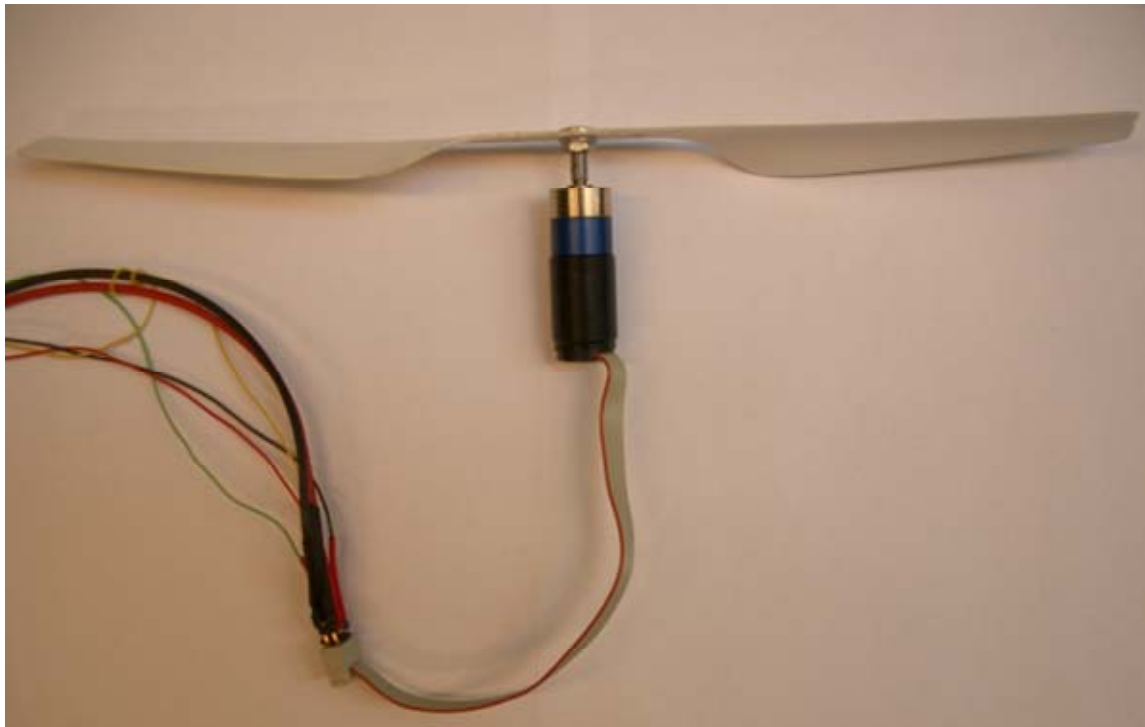


Figure 7.5: motor, reductor and encoder

Figure 7.5 shows the used motor, reductor, encoder and propeller.

7.6 Speed encoder

If the motor was set to turn at high speed (as in 7.5). Each channel would display a signal similar to the one on figure 7.6. Note that the frequency corresponds to about 28kHz (close to maximal frequency). It was shown that the period had to be averaged on 2 cycles.

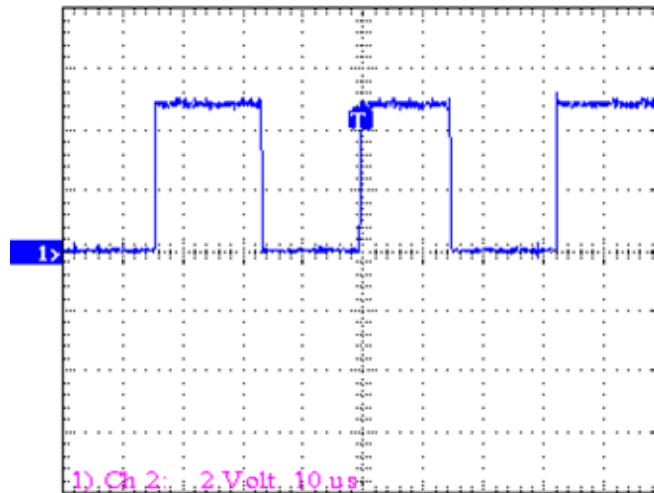


Figure 7.6 : channel A at maximal speed

7.7 Frequency doubler

Doubling the frequency is done by doing an XOR on the two speed encoder's channels. Since they are shifted by 90° the output of the XOR gate will be twice the input frequency. Figure 7.7 shows the input frequency on the top (only one channel is shown here, the second would be shifted by about 10uS) and the doubled frequency on the bottom. Note that averaging over 4 periods or more is required to have a precise reading of the frequency.

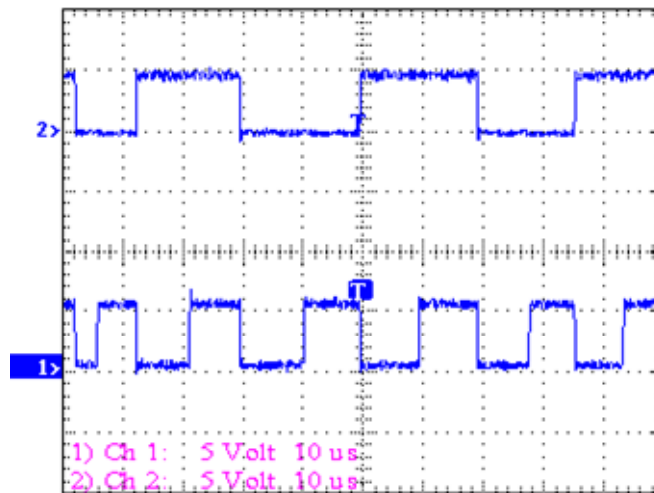


Figure 7.7 : frequency doubling

7.8 Frequency to voltage

As seen in section 4.9 converting a frequency to a voltage is not that trivial. For this reason tests were performed on a Hirschman board first and then on a PCB.

7.8.1 Using a Hirschman board

Here we assumed that the right motor would be used, which means that $R_1 = 35\text{k}\Omega$, $C_1 = 3.69\text{nF}$, $C_2 = 0.84\mu\text{F}$ (see section 4.8 for frequency = 3200Hz). These values should result in a 20mV noise and a 30ms setting time (step input).

Recall that in the final version R_1 will be around 27k Ω but in series with a 20k Ω resistor in order to allow a fine tuning.

Remember that the output voltage is equal given by: $V_{out} = V_{sup} \cdot R_1 \cdot C_1 \cdot f_{in}$, where $V_{sup} = 15$.

Thus for $f = 2600\text{Hz}$ (bottom of figure 7.8.1a) the output voltage should be 5V (top of figure 7.8.1a), which is the case on figure 7.8.1a.

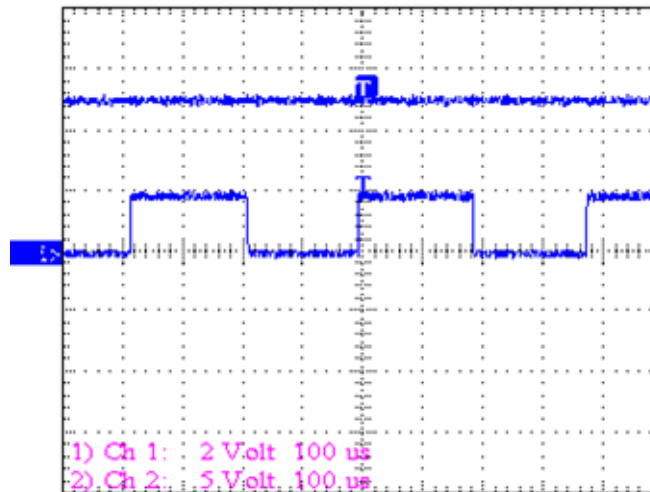


Figure 7.8.1a : output at 2600 Hz

Figure 7.8.1b shows that the noise level is around 20mV.

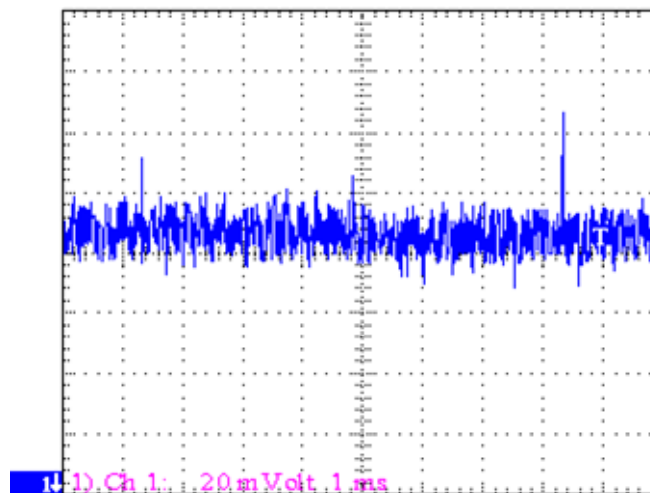


Figure 7.8.1b : noise level

Figure 7.8.1c shows the output if a frequency jump from 300 to 3000 Hz is performed. One can observe the time constant τ of about 30ms. And a settling time of also 30 to 40 ms (settling time: go from 10% to 90% of final value).

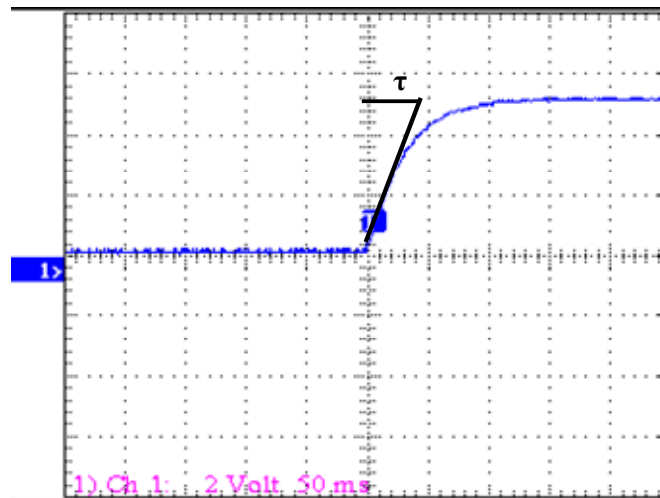


Figure 7.8.1c : frequency jump

7.8.2 On the PCB

The same kind of tests were performed on the PCB version.

Figure 7.8.2a shows a frequency jump from 3.2kHz to 32kHz (maximal frequency of the test motor). Again the time constant and rise time are in the expected range.

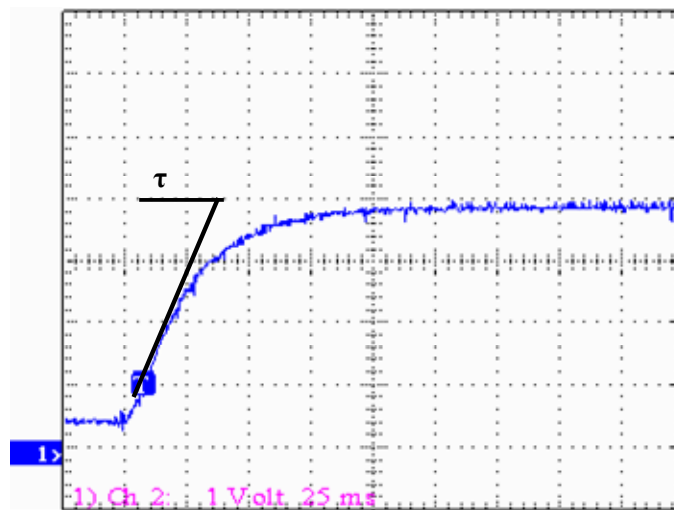


Figure 7.8.2a : frequency jump

Figure 7.8.2b shows the noise level, which is higher than expected. So far it was impossible to determine where the problem was. SMD components were checked however no bug was found. It is thought that this comes from the fact that C_1 is too small, however this won't be an issue anymore when the right motor will be used.

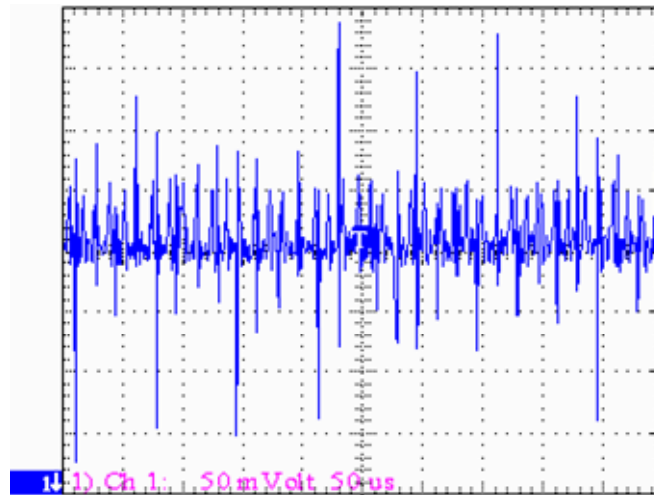


Figure 7.8.2b : noise level

Figure 7.8.2c shows what the voltage is as a function of the DAC's input. One can observe a slight nonlinearity as well as a non-zero value of the output voltage even though f_{in} is zero. These problems come from the fact that C_1 is only 388pF and should be at least 500pF. But due to the motor constraints this can't be changed. However when the real motor will be used C_1 will be way above this limiting value, so these problems should vanish.

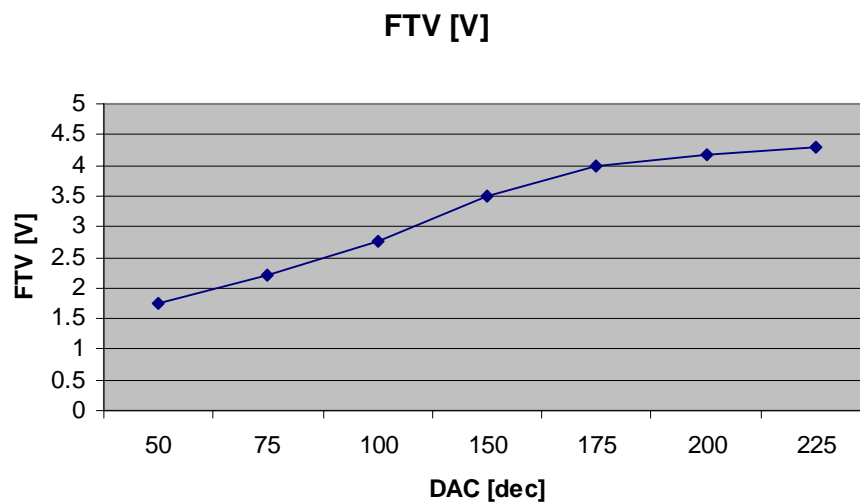


Figure 7.8.2c : FTV output as function of DAC input

Finally figure 7.8.2d shows how this nonlinearity problem was solved. Knowing that the motor would be used between 70 and 100% of its maximal velocity one managed to fit the real FTV output voltage and the DAC output voltage (command). One can observe that for DAC inputs of 140 or higher the matching of the two curves is good.

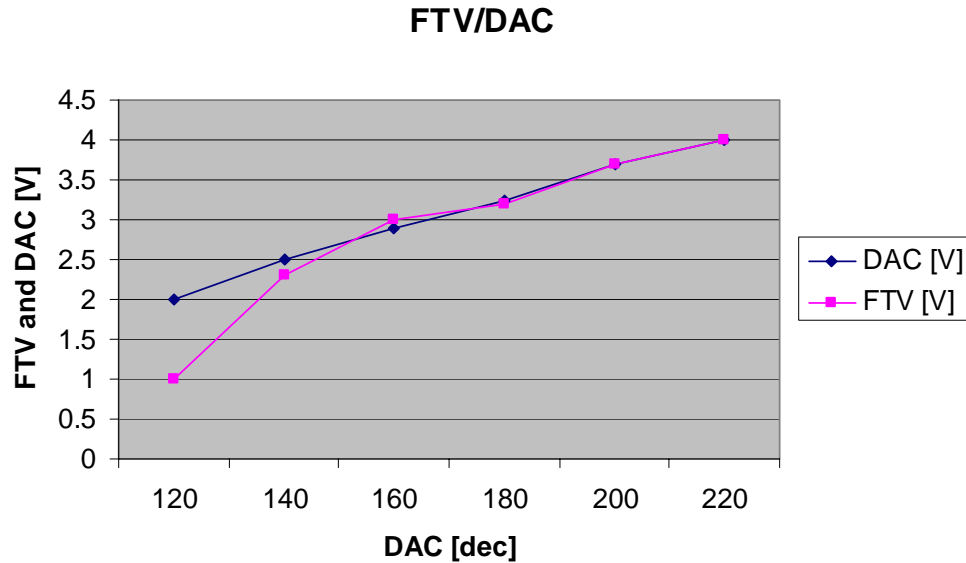


Figure 7.8.2d : FTV output as a function of DAC input

7.9 Butterworth filter

The filter will be used to smoothen the frequency to voltage output without using the internal (C_2) capacitor. Doing this allows to have faster setting time.

7.9.1 Using a Hirschman board

Using the values of resistors and capacitors calculated in 4.10 one should find a time constant equal to 3 to 4 ms. As can be seen on figure 7.9.1 if a square signal is applied at the entrance of the filter this rise time is observed.

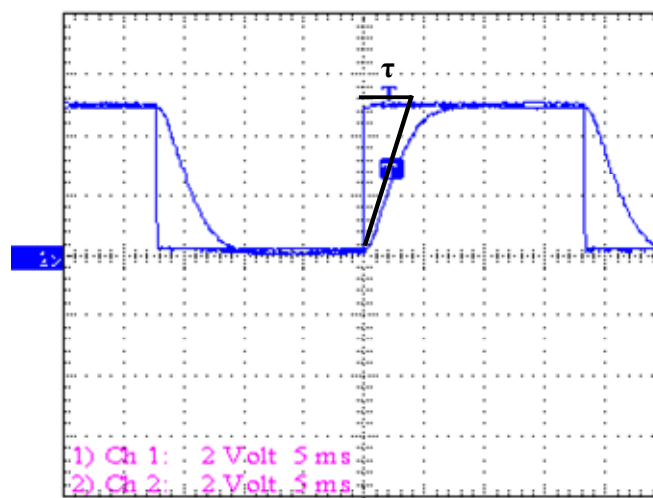


Figure 7.9.1 : filter step response

Using a dBmeter showed that the cut frequency was slightly higher than the 100Hz that were expected, however at around 200Hz the signal was already extremely weak.

7.9.2 On the PCB

The same experiment as in section 7.8.1 was performed on the PCB except that this time the C2 capacitor of the frequency to voltage was replaced by the Butterworth filter. One can notice that first the signal is increasing slowly for 2.5ms, most probably because of an internal capacitor in the FTV converter. But after this short time a time constant of around 4 to 5 ms is found, as seen in 7.9.1a.

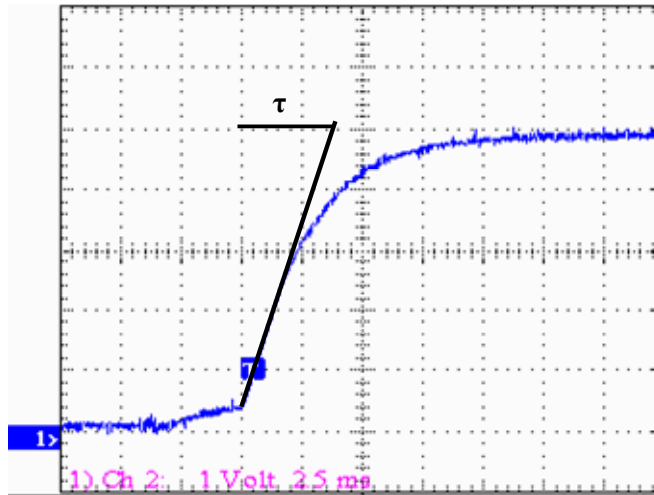


Figure 7.9.2a : filter step response

Figure 7.9.2b shows the noise at the filters output. As seen in 7.8.2 its amplitude is twice the one we expected, as explained before this is probably due to the fact that C1 is too small.

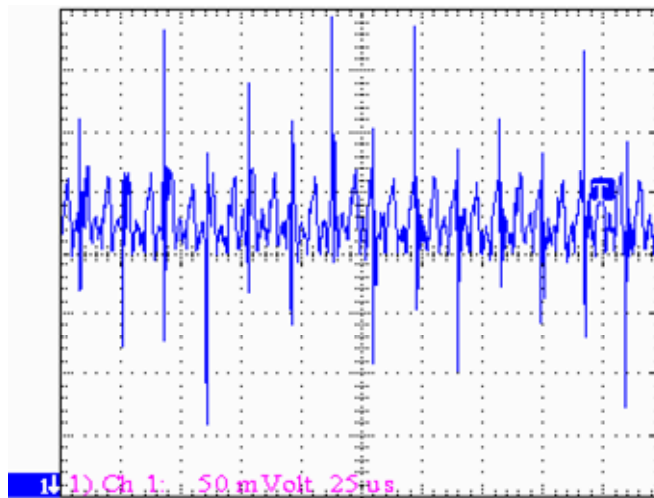


Figure 7.9.2b : filter noise

Figure 7.9.2c shows the unfiltered output. Some simple math show that the filter is doing a 20dB attenuation ($20 \cdot \log\left(\frac{\text{Unfiltered_Noise}}{\text{Filtered_Noise}}\right)$ where unfiltered noise = 2.5V and filtered noise = 250mV).

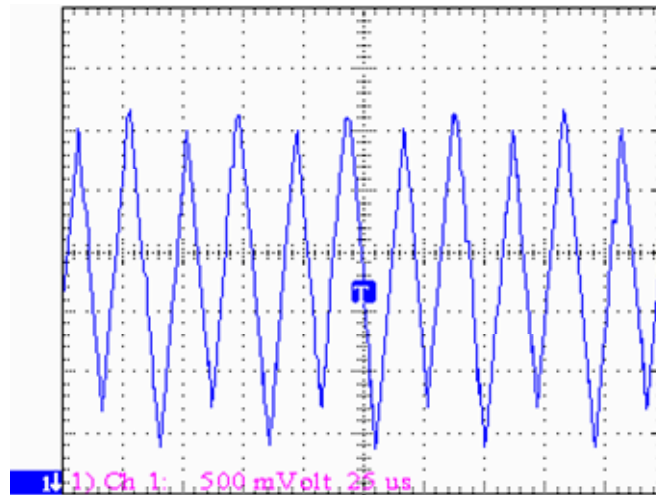


Figure 7.9.2c : unfiltered answer

7.10 Consumption

Item	Consumption	Comment
Speed encoder	10mA at 5V	
LM 2907	10mA at 12 V	Converted from 5V
Rest of the PCB	150mA at 5V	
Total	160mA at 5V	

Measuring the consumption of the PID PCB is not an easy task since most components work at the same time. However for the removable components the consumption is given in the table above.

The table below gives the consumption of the PCB if its alimented only in +12V, -12V or +5V.

Alimentation	Consumption
5V	10mA at 5V
+12V	50mA at 5V
-12V	70mA at 5V
Total	140mA at 5V

Note that in the table above the total consumption is only 140mA whereas the previous table indicated 160mA. This comes from the fact that some components are interconnected and use different voltages to work. And if one is not functioning then the second even though it gets a supply voltage won't do anything.

8 Open loop (without PID)

In order to illustrate the need for a closed loop PID control the following test was made. The motor was asked to go from 0rpm to a speed corresponding to a DAC input of 220. The result can be observed on figure 8.1a, where the upper signal is the command and the lower curve the read speed.

Several conclusions can be drawn. First the rise time is sort of high but worse there is a steady state error.

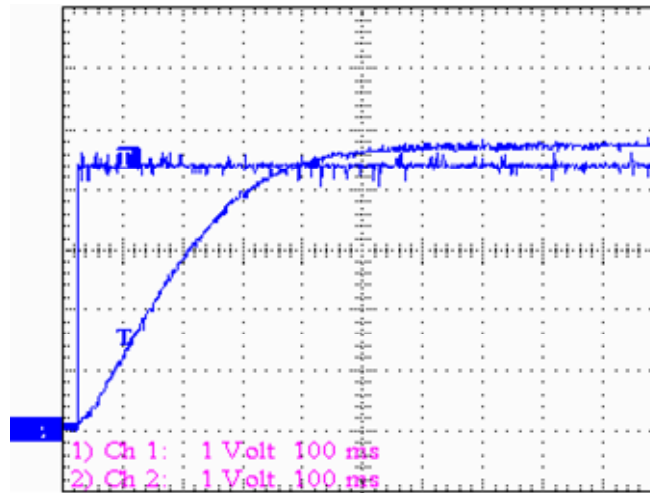


Figure 8.1a : open loop response

Figure 8.1b shows the simulated behaviour, one can observe that they match. Also note the FTV's saturation at about 4.7 V on figure 8.1b.

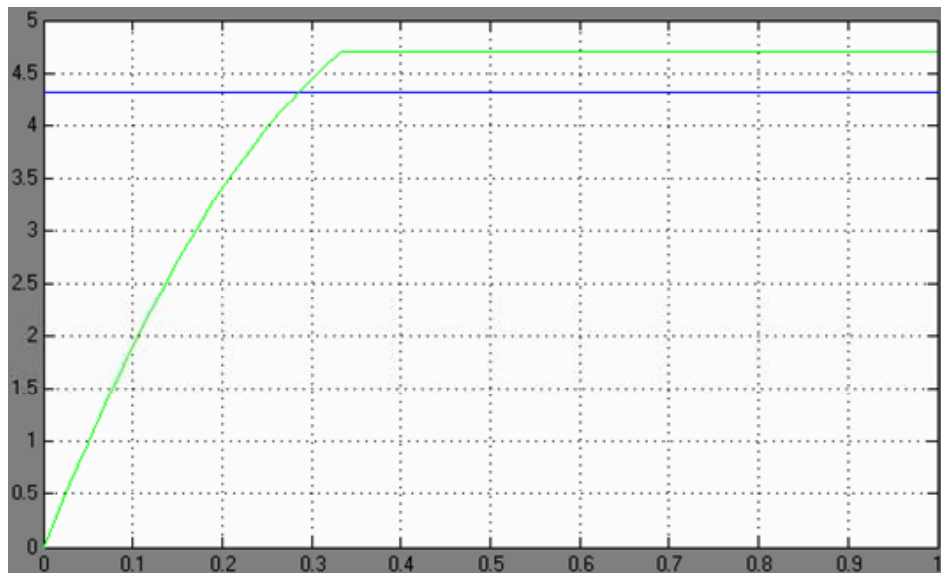


Figure 8.1b : open loop (simulated)

Figure 8.1c shows that the transfer function calculated in section 5.5 is correct. Indeed plotting the measured motor rpm and the calculated motor rpm give almost the same curve.

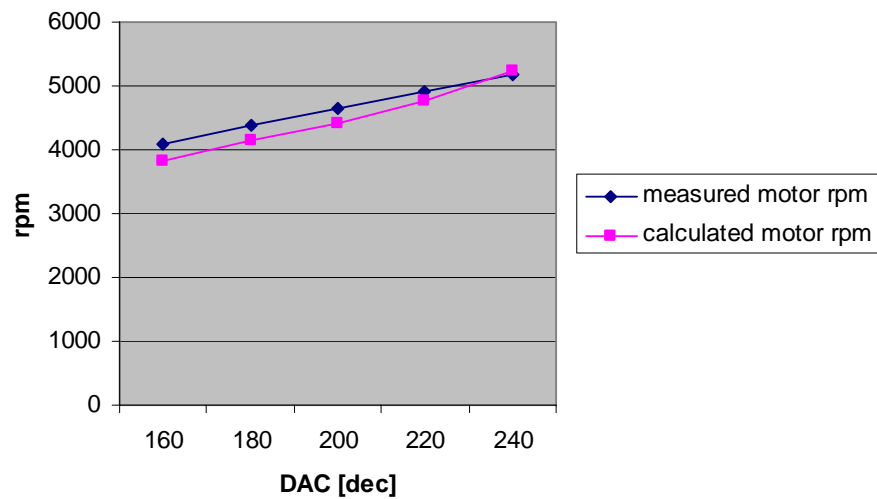


Figure 8.1c : measured and calculated rpm in open-loop

Figure 8.1d shows how the error between the measured and calculated curve is changing as a function of the DAC input.

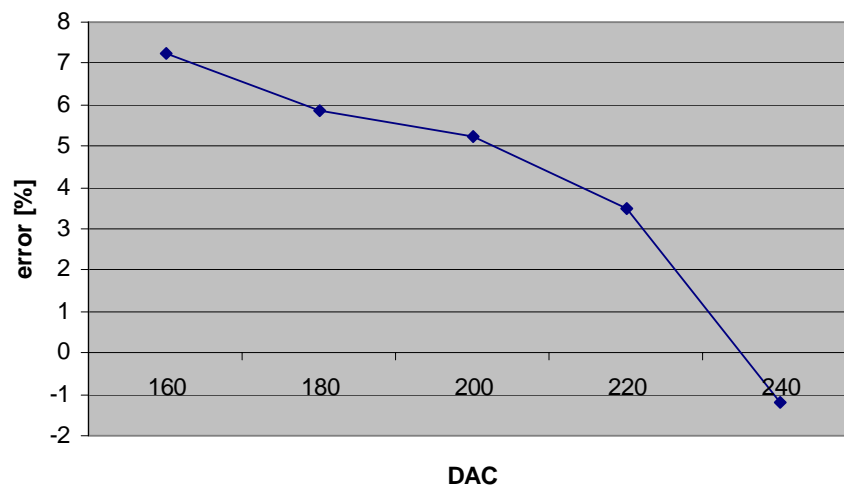


Figure 8.1d: open-loop error

9 Closed loop: K_p only

First tests were done where K_i and K_d were set to zero and only K_p was considered. This was mostly done to ensure the correctness of the different signals.

9.1 error = command – speed ($K_p = 1$)

As in chapter 8 the motor was requested to go from rpm to a speed corresponding to a DAC value of 220.

Except that this time closed loop operation was performed and K_p was equal to 1. This is simply done by setting $R_1 = 5k\Omega$, $R_2 = 5k\Omega$ and shorting C_2 .

Figure 9.1a shows two curves. The upper one is the command signal and the lower one the read speed. Note that the desired speed is never reached and thus a big steady state error exists. The steady state consumption is around 200mA at 8V.

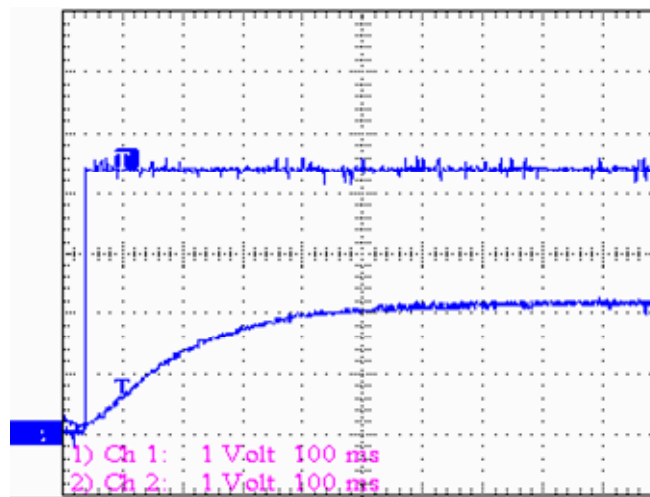
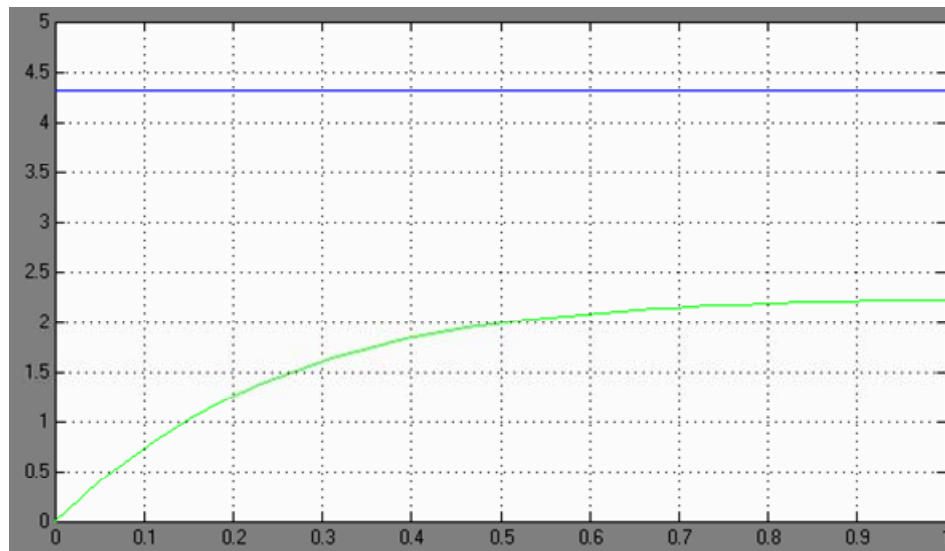


Figure 9.1a : closed loop response ($K_p = 1$)

Figure 9.1b shows the simulated values, one can observe that they match the values of figure 9.1a.

Figure 9.1b : closed loop response simulink ($K_p = 1$)

9.2 $K_p = 6$

The same experiment as in 9.1 was done, but this time with $R_1 = 5k\Omega$, $R_2 = 30k\Omega$. Thus $K_p = 6$.

One can observe the measured values in figure 9.2a and the simulated ones in figure 9.2b. Again they match. The consumption is about 300mA during the acceleration phase and 200mA in steady state.

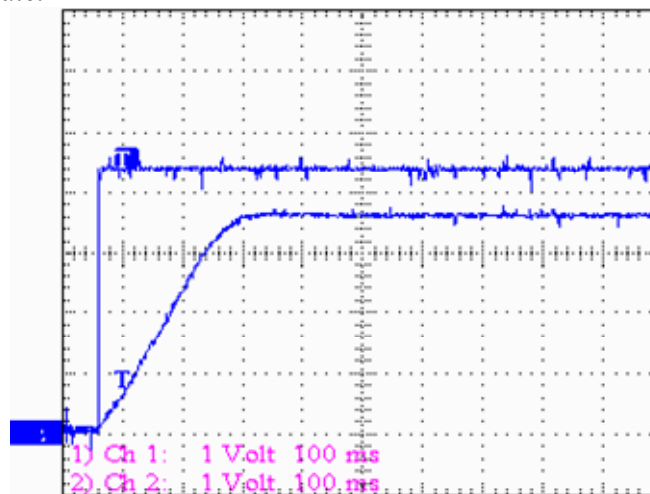
Figure 9.2a : closed loop response ($K_p = 6$)



Figure 9.2b : closed loop response simulink ($K_p = 6$)

10 Closed loop with PID

In order to show that the PID controller was working several measures were done.

10.1 Non optimal PID values

By choosing $R_1 = 100\,000\Omega$, $R_2 = 470\,000\Omega$, $C_1 = 5\text{nF}$ and $C_2 = 5\mu\text{F}$ one gets:

$$K_p = 5, K_i = 2, K_d = 0.00082$$

Under these conditions the motor was asked to go from 0rpm to a speed corresponding to a DAC value of 220.

The upper signal of figure 10.1a is the command signal and the lower is the read speed.

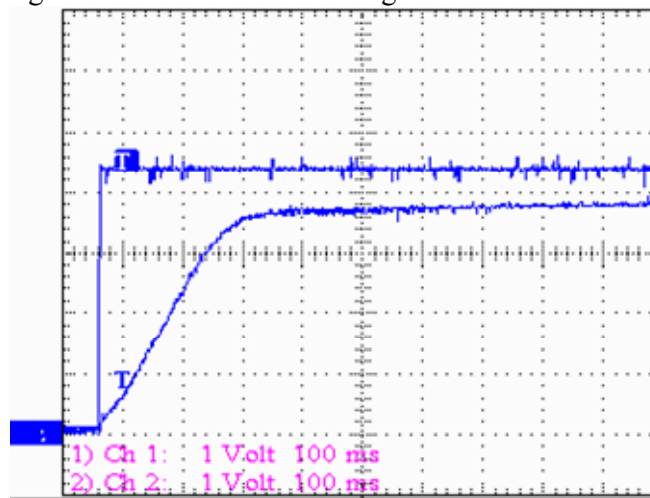


Figure 10.1a : closed loop response (PID enabled)

Figure 10.1b shows the simulated values, note that they match the reality. Consumption is around 500mA during acceleration and 350mA in steady state.

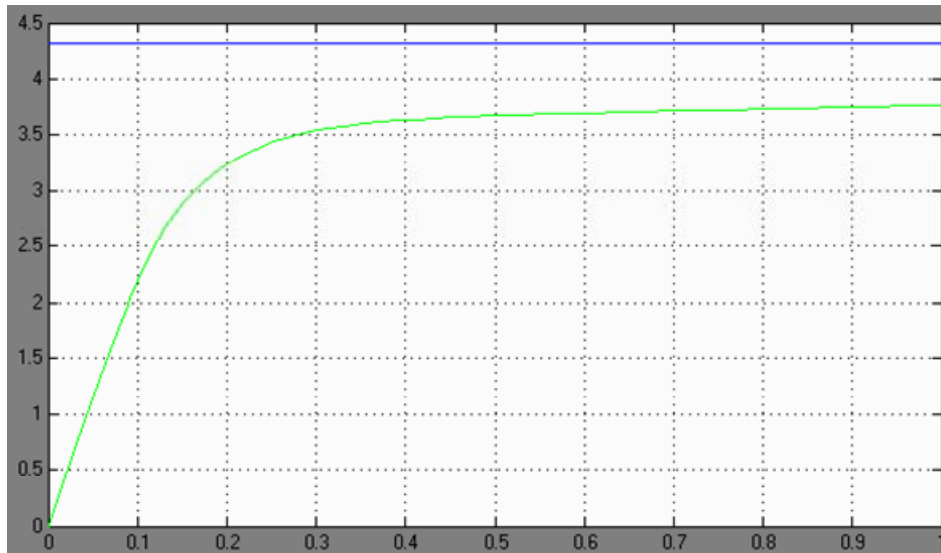


Figure 10.1b : closed loop response simulink (PID)

As expected for non optimal PID values the rise time is not amazingly good nor is the steady state error

10.2 Better values from Simulink

Using simulink one can find better PID values.
 $K_p = 8$, $K_i = 11$, $K_d = 0.001$.

By choosing $R_1 = 100\,000\,\Omega$, $R_2 = 820\,000\,\Omega$, $C_1 = 1\text{ nF}$ and $C_2 = 1\text{ }\mu\text{F}$ one gets:

$K_p = 8.2$, $K_i = 10$, $K_d = 0.00082$

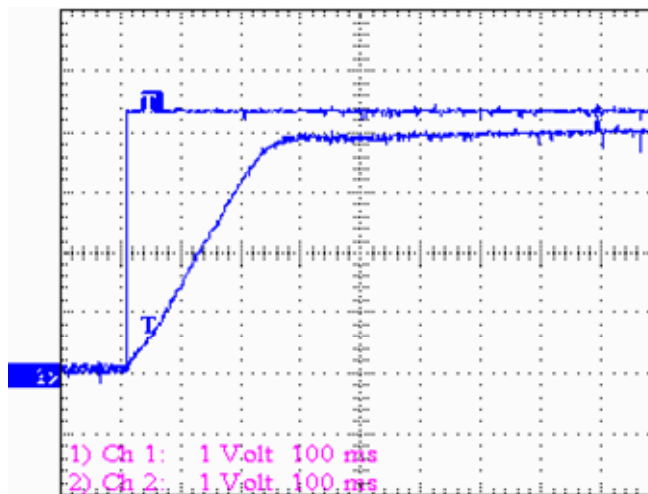


Figure 10.2a : closed loop response (PID enabled)

The upper signal on figure 10.2a is the command and the lower is the read speed, one can observe that the speed increase is almost linear. Settling time and steady state error are better than before.

Again figure 10.2b shows the simulated value, observe that they match the reality.

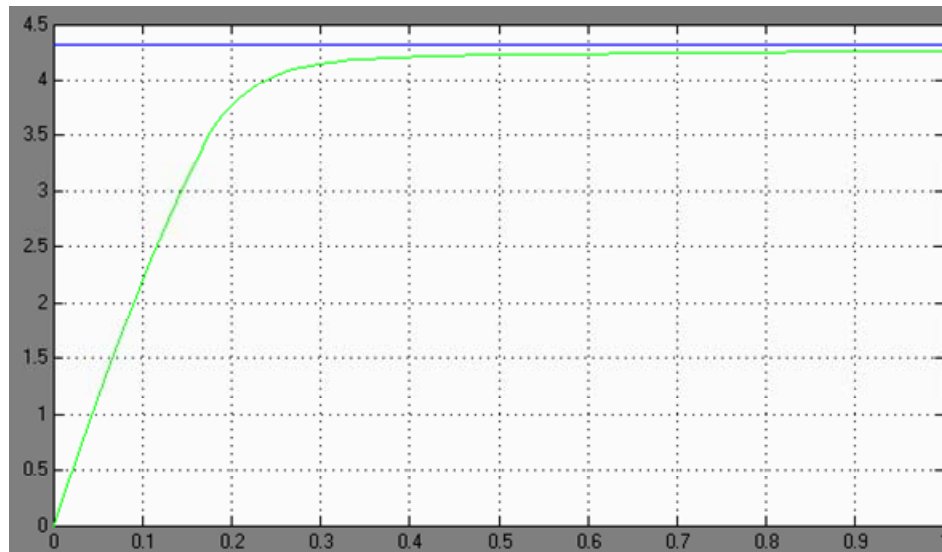


Figure 10.2b : closed loop response simulin (PID)

10.3 Overshoot

Recall that the H-bridge can operate in two modes. Basically the reverse can be switched off and replaced by a short of the motor input, which leads to a slow decay. This effect will be illustrated now.

In order to have a nice overshoot the following values are wanted:

$$K_p = 2, K_i = 20, K_d = 0.001$$

By choosing $R_1 = 100\,000\,\Omega$, $R_2 = 180\,000\,\Omega$, $C_1 = 5\text{nF}$ and $C_2 = 470\text{nF}$ one gets:

$$K_p = 1.8, K_i = 21, K_d = 0.00084$$

10.3.1 Reverse disabled

In this case the reverse was disabled.

On figure 10.3.1a the step signal is the command and the other is the speed, note the nice overshoot.

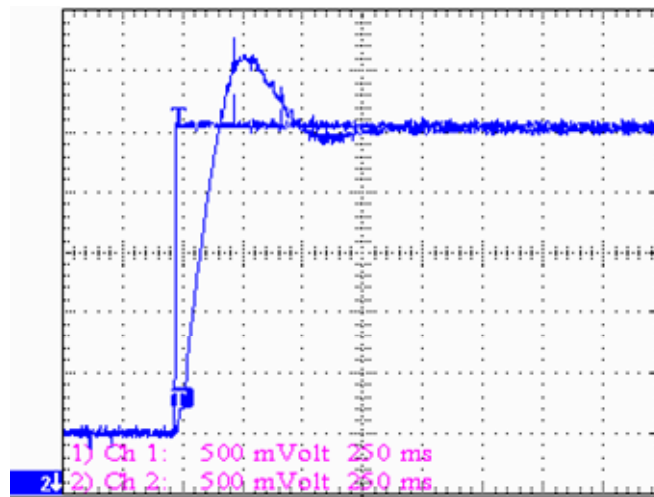


Figure 10.3.1a : closed loop response (PID enabled)

Figure 10.3.1b shows the simulated values.

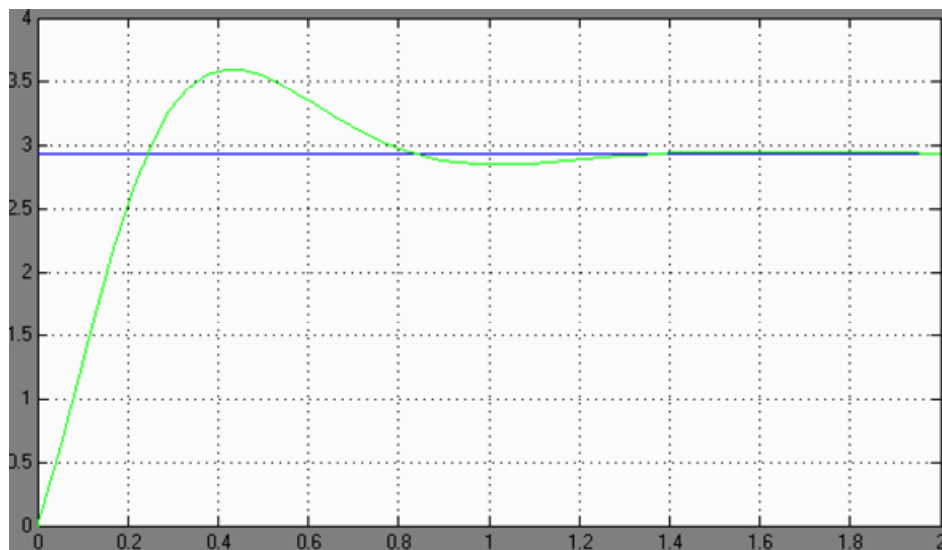


Figure 10.3.1b : closed loop response simulink (PID)

10.3.2 Reverse enabled

In this case the reverse was enabled.

On figure 10.3.1b the step signal is the command and the other is the speed.

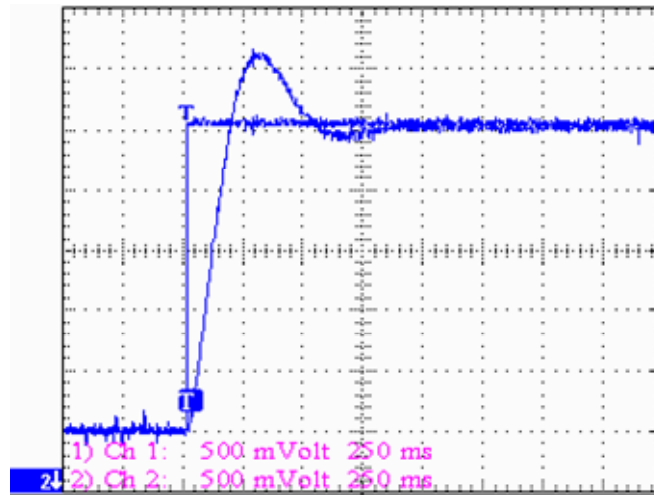


Figure 10.3.2a : closed loop response (PID enabled)

Note that if reverse is disabled the distance between the peak overshoot and the part where the speed goes up again is slightly bigger as in the case where reverse is enabled. This is what we expected.

10.4 Optimal PID values

By choosing $R_1 = 100000 \Omega$, $R_2 = 1200000 \Omega$, $C_1 = 830 \text{pF}$ and $C_2 = 220 \text{nF}$ one gets:

$$K_p = 12, K_i = 42, K_d = 0.001$$

Under these conditions the motor was asked to go from 0rpm to a speed corresponding to a DAC value of 220.

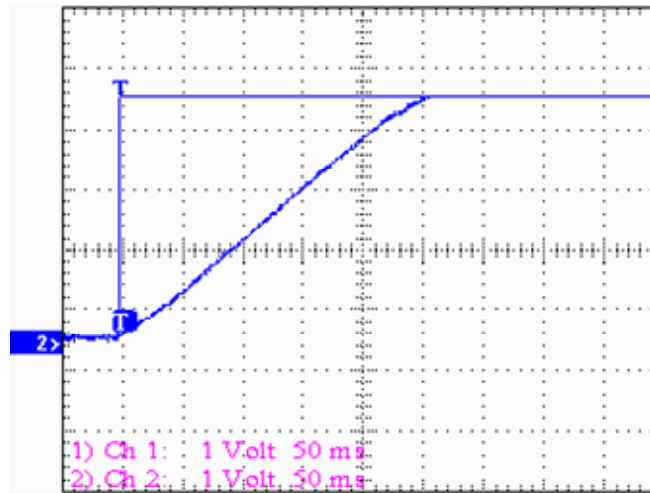


Figure 10.4a : closed loop response (PID enabled)

Observe that as expected increasing K_i helps to get a faster setting time. However in this case the simulation indicates an overshoot whereas tests clearly indicate that there is none. We believe that this is due to the fact that our simulation model is quite simple and maybe doesn't take into account all the parameters and is thus not able to deal with so "aggressive" PID values.

10.5 Tachymeter test

In order to prove that the motor was turning at the requested speed further tests were done using an infrared tachymeter.

The following table summarizes the read values. Note that the calculated rpm has to be multiplied by 2, because there are two blades on the propeller. And there is also a reduction of 3.71 between the motor and the propeller.

DAC	read propeller rpm	simulink propeller rpm	error %
60	546	582.00	6.19
80	736	782.00	5.88
100	930	977.00	4.81
120	1113	1173.00	5.12
140	1296	1369.00	5.33
160	1487	1564.00	4.92
180	1683	1760.00	4.38
200	1874	1956.00	4.19
220	2072	2152.00	3.72

One can observe that for low speeds the error gets worse, but is constant otherwise. Refer to figure 10.5a for a better representation.

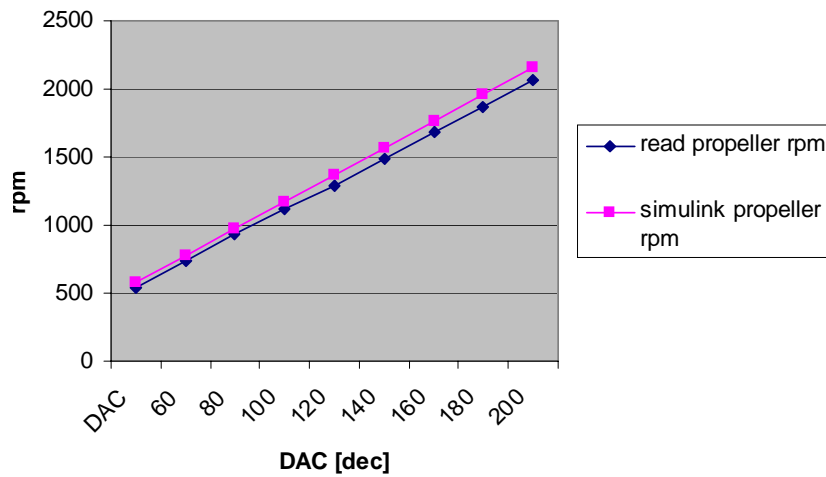


Figure 10.5a: theoretical and measured rpm

Since the helicopter only works at high speed these errors should not be an issue, also maybe the tachometer was not well calibrated.

One can also use the following transfer function to get the output speed given a certain DAC value.

$$RPM_{motor} = \frac{dec_value \cdot V_{ref} \cdot 60}{255 \cdot R_1 \cdot C_1 \cdot N \cdot V_{sup}}$$

Where:

V_{ref} is the DAC reference voltage.

C_1 and R_1 are given by the FTV converter.

N is the number of increments of the speed sensor.

Dec_value is the decimal value that is sent to the DAC.

Then converting from motor rpm to propeller rpm simply means dividing the former by 3.71. Pay attention to the fact that there are two blades.

The following table is obtained if the above equation is used.

DAC	calculated propeller rpm	read propeller rpm	error %
60	572	546	4.7
80	762	736	3.6
100	953	930	2.4
120	1143	1113	2.7
140	1334	1296	2.9
160	1524	1487	2.5
180	1715	1683	1.9
200	1905	1874	1.7
220	2096	2072	1.2

Plotting it gives figure 10.5b.

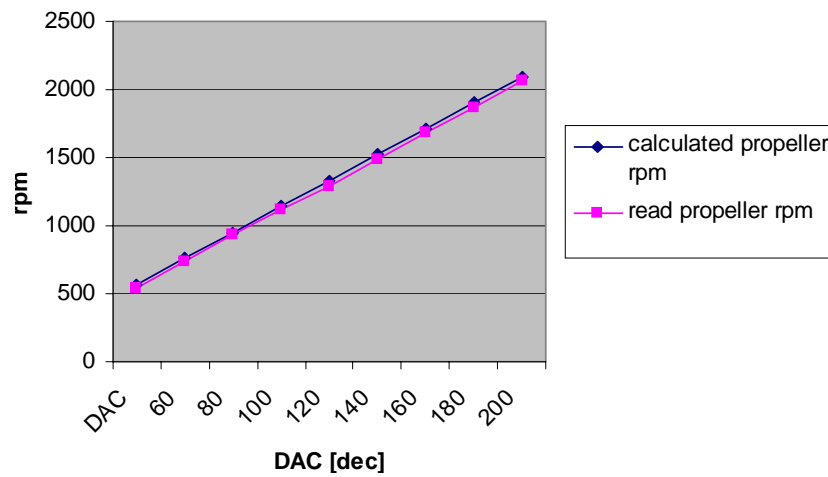


Figure 10.5b: theoretical and measured rpm

In conclusion, the transfer function can be used to calculate the final closed loop speed, it will however assume that the motor will turn at the commanded speed. Or simulink can be used, in this case the motor dynamics are better modelled. In both cases the calculated value is slightly above the practical one. However in the operation range this error is constant and quite small.

11 Conclusion

11.1 General conclusion

One of the first times I met Samir I asked him: “What is hard about this project?”. Obviously my ideas on hardware and control were mainly theoretic. It turned out that developing a fully operational PID module was quite a delicate task. To prove this one simply needs to look at what my PCBs look like after some heavy bonding was done.

However we achieved to obtain a fully analog module, that is commanded as if it was a digital module, which regulates the speed as expected. It was thus shown that building an analog speed controller was feasible and that it could be suited for standard applications. Further more we have the feeling that further investigations should be done which could lead to a new standard PID controller module.

In order to illustrate that the given module is working it was used to optimise the response time of a motor that is comparable to the one available one the OS4, and it worked well.

On a personal level I have gained valuable knowledge mostly in the analog and automotive field. I also learned to use Protel and familiarized myself with the process of getting a PCB done.

Yves Stauffer

11.2 Are the goals met?

Section 2.1 gave guidelines for this project. How close are the obtained modules to the desired specifications?

- 1) The response time is as fast as analog can provide, the FTV being the limiting part.
- 2) Lightness and volume can be quite reduced; the used version is heavy mainly because lot's of debugging had to be done.
- 3) On the central PCB only two outputs are available, however adding two more is easy.
- 4) The communication protocols are respected.
- 5) The H-bridge can deliver up to 2A and can work with voltages as low as 6.2V.
- 6) The controller gains can't be changed dynamically of course, but jumpers allow a quite fast tuning.
- 7) The resolution is mainly limited by the DAC and FTV noise levels, theoretically the achievable resolution is around 40mV, which corresponds to 35 rpm (referenced to the motor's output) increments. Immeasurable with the available equipment.

One can see that the goals are met.

11.3 What can be improved

The main problem is the lack of direction of rotation sensor. This is especially dramatic since the real motor was not used for the tests. And because of the small offset in the FTV conversion the error signal was different from zero even though the command signal and speed was zero. This resulted in highly unstable behaviour.

Finally, if wanted a new semester project can be proposed where my design could be upgraded to an FPAA PID.

11.4 Special thanks to

Samir Bouabdallah: for proposing this project, his design advices and availability.

Daniel Burnier: for hardware advices, soldering help and debugging.

Gorges Vaucher: for the PCB routing and overall patience.

Jacques Fournier: for his filter analysis.

Andre Decurnex: for his advices on OA.

Andre Noth: for the modelling part.

Roland Siegwart: for accepting this challenging project.

Allegro, TI and Maxim for the samples.

12 Bibliography

[1] Niese, *control systems engineering*, 2000

[2] Phillips and Harbor, *Feedback Control Systems*, 2000

[3] Andre Noth, *Synthese et Implementation d'un Controleur pour Micro Helicoptere a 4 Rotors*, 2004

13 Annexes

13.1 Protel schematics

13.1.1 Original Central PCB

13.1.2 New Central PCB

13.1.3 Original PID PCB

13.1.4 New PID PCB

13.2 ACORT files

13.2.1 Central PCB files

13.2.2 PID PCB files

13.3 Relevant data sheets (extracts)

13.3.1 DAC: MAX 520

13.3.2 H-Bridge: A 3949

13.3.3 Frequency to Voltage converter: LM 2907

13.3.4 Motor

13.3.5 Speed encoder

13.3.6 Reductor

<p>Note: in order to spare paper annexes 13.1 to 13.3 are regrouped in one block</p>

13.4 Mathematica code

```
Solve[{(R2/R1 + C1/C2) == 5, (1/(R1*C2)) == 10, R2*C1 == 0.001, R1 == 100000}, {R1, R2, C1, C2}] //KpKiKd
```

13.5 PCB pictures

13.5.1 Central PCB front

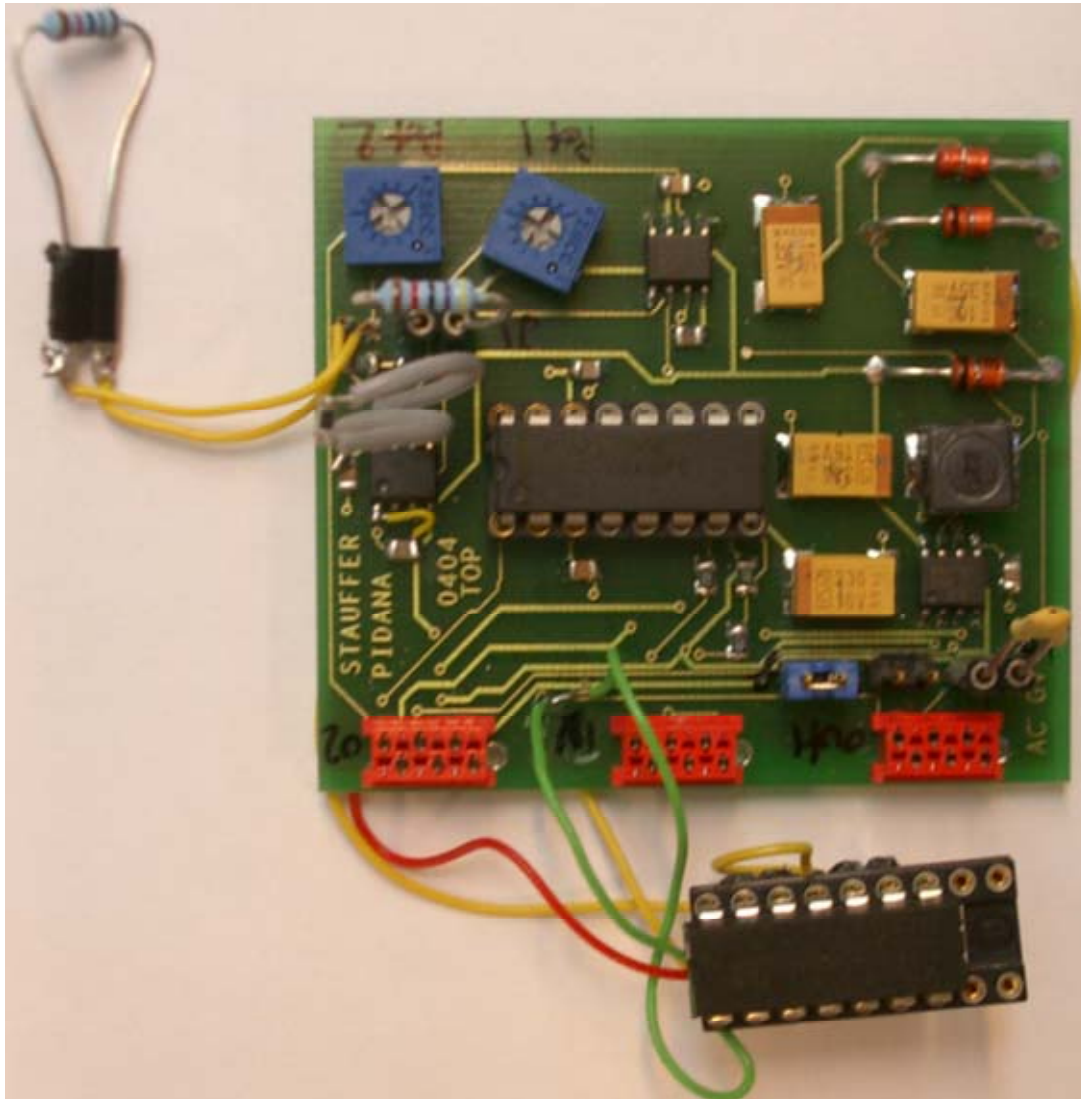


Figure 13.5.1: central PCB front view

13.5.2 Central PCB back

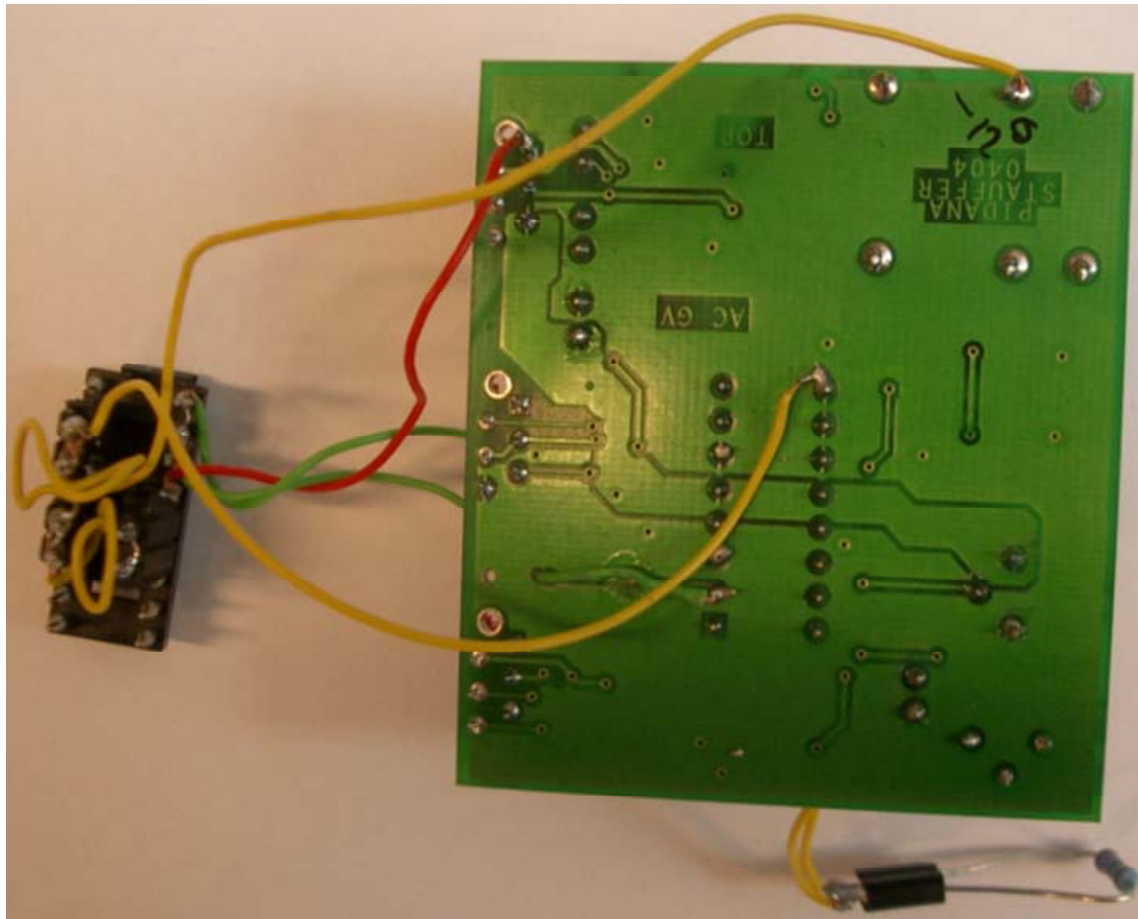


Figure 13.5.2: central PCB back view

13.5.3 PID PCB front

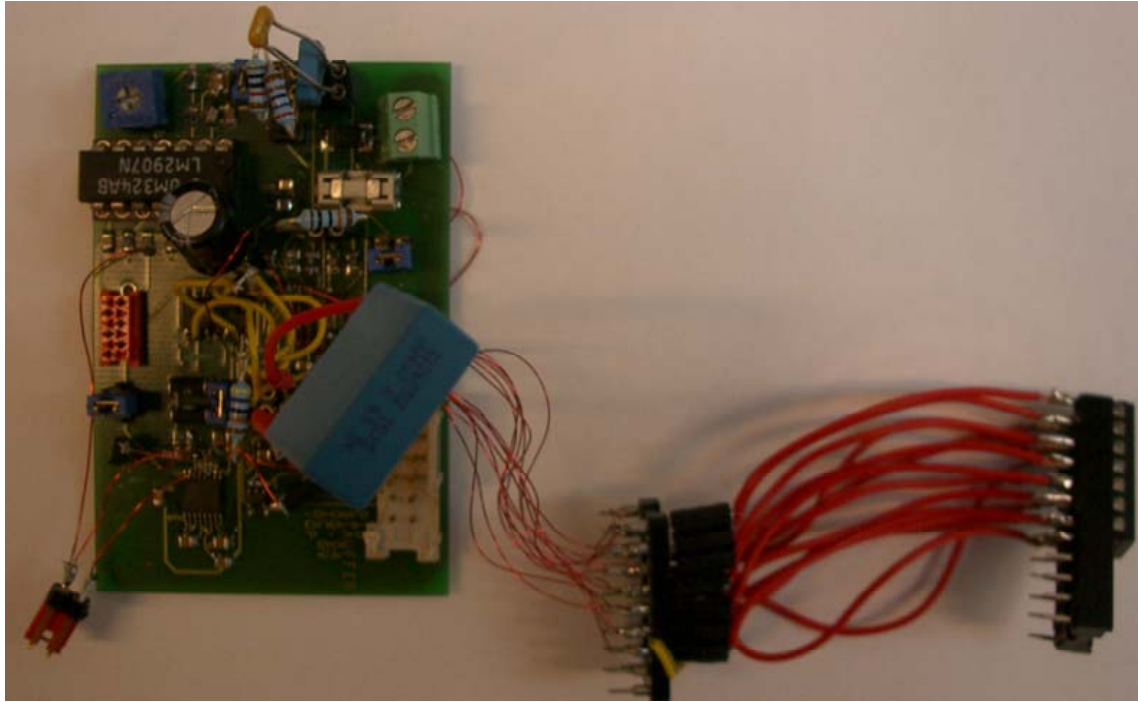


Figure 13.5.3: PID PCB front view

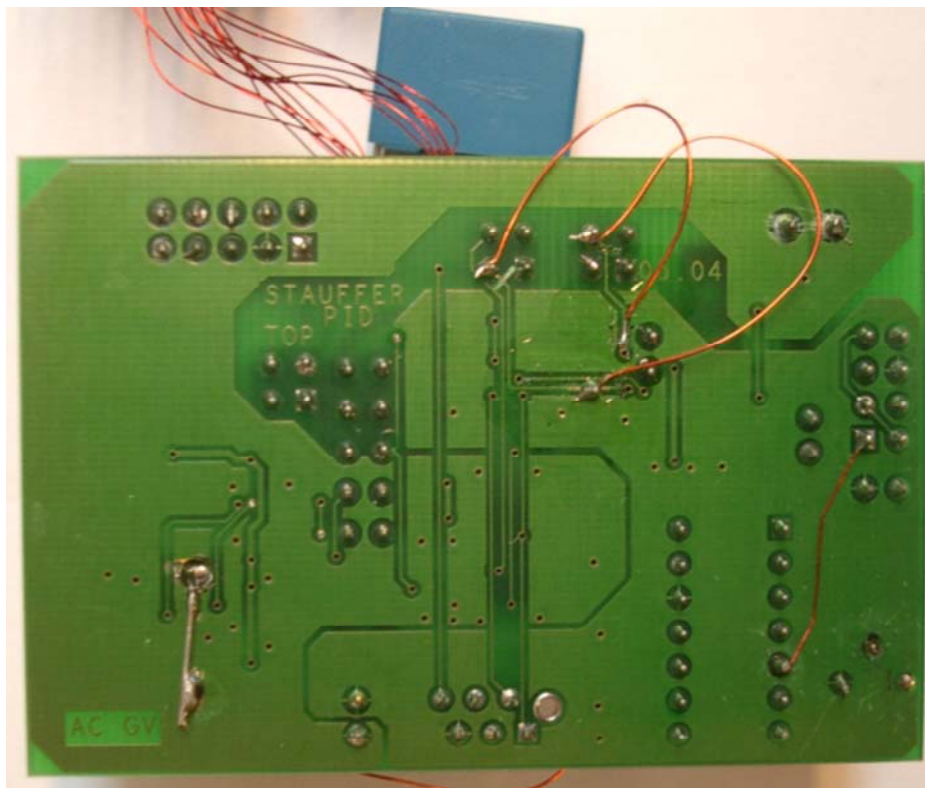
13.5.4) PID PCB back

Figure 13.5.4: PID PCB back view