MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
“IGOR SIKORSKYI KIEV POLYTECHNIC INSTITUTE”
THE DEPARTMENT OF DESIGN OF ELECTRONIC DIGITAL EQUIPMENT

COURSEWORK

in the discipline: Analog Electronics-2

on a topic: “DC motor PID controller”

student of the second course, DK-82 group
field of study: telecommunication and radiotechnics
Yehor Krapovnytskyi

Header:

_____associate professor, Korotkyi I.V._____

(position, academic title, academic degree, surname and initials)

National assessment: _____

number of points: ____ ECTS Assessment:_____

Member of a commission:_____ __associate professor, Korotkyi I.V.__

(signature) (position, academic title, academic degree, surname and initials)

Kiev-2020

TABLE OF CONTENTS

Introduction……2
List of acronyms……..3
1. Part 1 – Fundamentals of PID theory……..4
2. Part 2 – Working prototype description……..7
   2.1. List of used components……..7
   2.2. Hardware setup…….....8
   2.3. Software setup…….....10
3. Part 3 – Building-up a model……11
   3.1. Motor run and encoder read……11
   3.2. Determine the RPM…….12
   3.3. Plant model estimation……15
4. Part 4 – PID controller configuration and performance……..17
   4.1.PID controller design……17
   4.2. Implementing and testing PID controller on real hardware…….20

Conclusion……...23

Reference list……...24

INTRODUCTION

Assume that we have a system we want to control. The system has an input, where it receives a command signal and the output, where we can measure a controller variable and it must repeat the command one. Also, we have a feedback path that goes from the output, then difference between the input and the output is calculated to see how far the system is from the desired point, so we have a closed loop. If the requirements are not that strict, the engineer can use an open-loop system. It is not as accurate as the open-loop one, because the amount of input force is not adjusted based on the real position of the system, so it may deviate from the required direction, or may overshoot. Feedback allows the engineer to make adjustments by sensing the output and comparing

it with the input. If to speak about a system with feedback loop and a controller, we imply a very sophisticated system, on which depend many prime things like human health, economic power of the country or even lives. For such a system requirement are of the highest level, so it must operate ideally. To make it to operate in such way, the corresponding controller is required. There are different types of controllers such as PI,PID,PD,etc. The goal of my course works is to implement a PID controller for the nonlinear system. In my case, I will use a DC Motor as a nonlinear system. It is a good variant, because of its low cost and convenience in use with motor drivers, and, of course it is simply enough to begin studying control theory. Also, the course project is actual, because PID control system is widely used in everyday life, for example a climate control or cruise control of the vehicle. In addition, studying this topic requires basic knowledge in control theory, that is very useful for further electronic projects, or automotive engineering.

In earlier times PID controllers were made of operational amplifiers and were exceptionally analog. Now they are implemented by software inside the PC, microcontroller or FPGA. In the project I will use MATLAB to build a motor model and design a PID controller for it. MATLAB allows to automate the process and make it more accurate. This approach is modern and the best for the engineers.

LIST OF ACRONYMS

PID	Proportional, Integral, Derivative
PI	Proportional, Integral
DC	Direct current
PPR	Pulses per revolution
RPM	Rotations per minute
FPGA	Field-Programmable Gate Array
PD	Proportional, Derivative

PART 1

FUNDAMENTALS OF PID THEORY

During typing the introduction, I have mentioned a word “system” a lot of times. In control theory this system is called a plant and our engineering purpose is to implement a controller for the plant, which provides the highest fidelity, lowest time response and cost. In my case it will be a PID controller. It uses the difference from the input and the output and then provides appropriate gains for all of P ,I and D terms to get a desired value on the output of the plant in minimum amount of time. A simple block diagram of the system with PID controller is showed in figure 1:

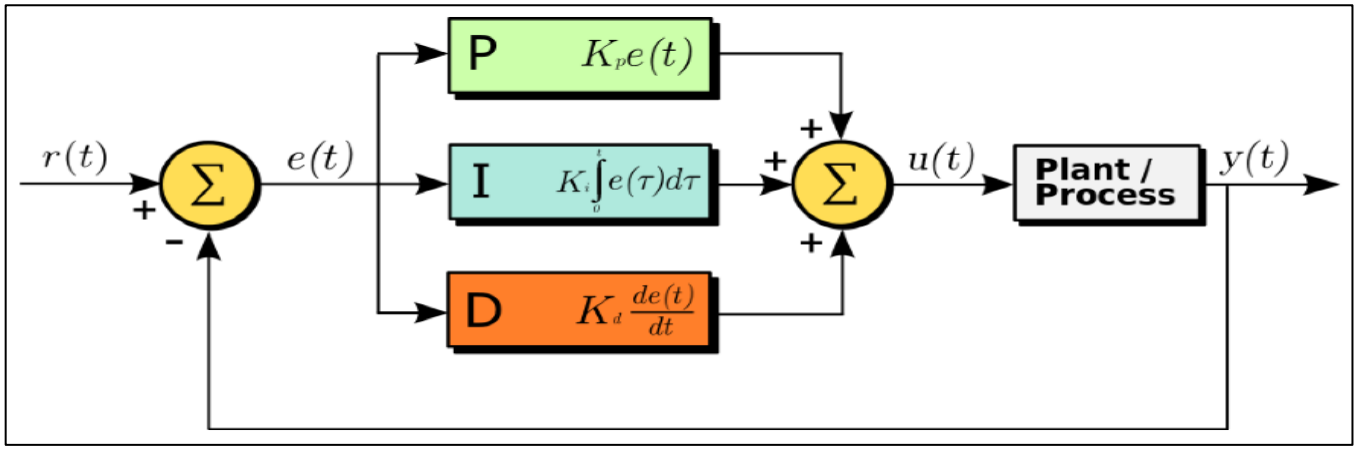

Figure 1: plant with feedback path and PID controller

Let’s skip over each of P, I and D terms in common words. Proportional part acts to reduce the error by simply amplifying or attenuating the error signal depending on its sign. The Derivative part is used to predict the system state and depends on the rate of the error signal change. If the error decreases too fast (that meant that P term is too high), then the derivative is negative and it is used to dampen the resulting signal to prevent overshooting. The Integral part of the controller is used to take down a steady state error. Assume that our system is a little out of desired point, but the error is too low to make the proportional part act. The small error accumulates in time and makes the Integral part to provide the finish touch to get 100% fidelity. After passing through each term the signal is summed and fed to the plant.

The main issue is how to adjust each of P, I and D terms to get the desired result. That’s why a PID Tuning plan exists and I have learning it via Brian Douglas videos on YouTube. It is shown in figure 2 as a very simple block-scheme. Obviously, I will use the simplest tuning methods because they cover the whole fundamentals and are very important if to speak about studying control theory.

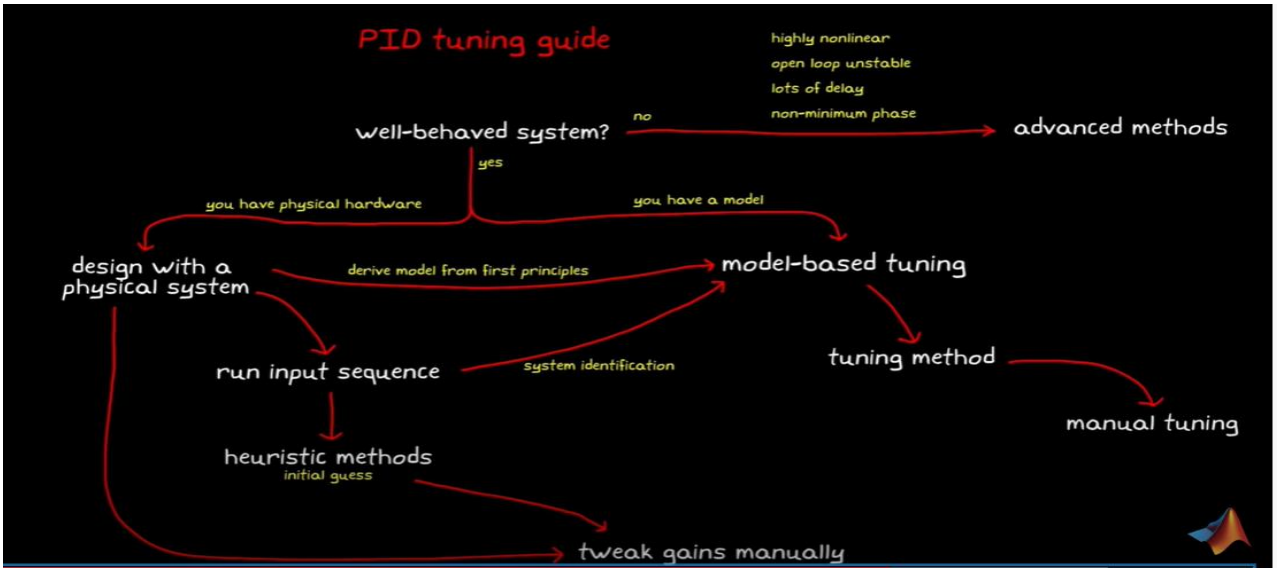

Figure 2: PID tuning guide block-scheme

The best approach for designing a PID controller is to represent a real model as a transfer function and test it on the software: in my case MATLAB Simulink. This will prevent a physical hardware to collapse during tests. Then, I will design a controller for the model and then test it on real hardware. This way is nice because it combines both efficiency and working with software to get accurate result.

By the way there are 3 ways to build a model:

- Derive from first principles
- Linearization
- System Identification Tool

I assume that my DC motor is well-behaved, and that's wright of comparing with complex systems. As I said above, I have a physical hardware. I could derive a model from first principles, but as an electrical engineer I'm not really interested in DC motor mechanics and differential equations that describe it. I will measure input and the output of the plant and then use System Identification tool to represent a motor as a second-order transfer function. After I have done this, I will apply manual tuning and MATLAB PID Tuner to get acceptable P, I and D terms, that control the system.

For this course work my plan is:

Design a physical system → run input sequence → model-based tuning → manual tuning/MATLAB Tuner

After doing this I will compare each of variants above and make a conclusion.

A plant with a PID controller can be represented as the next transfer function:

$$\frac{G_p * G_c}{1 + G_p * G_c}$$

Where: Gp — plant transfer function

Gc — PID controller transfer function

In this coursework I have skipped most of control theory maths, because it is sophisticated and doesn't represent the whole real situation and may be useful only in theory. Of course, I can calculate appropriate terms and spend a lot of time for that, but it doesn't guarantee the best result and in practice can be useless. A simply try and error method is more useful and convenient in this case. It requires less time and flexible enough to tune a PID controller.

PART 2

WORKING PROTOTYPE DESCRIPTION

2.1 List of used components (table 1)

№	Title	Description
1	L298N	Dual H-bridge motor driver. Technical parameters: • Power-supply voltage: DC 5-35 V • Peak current: 2 Amp • Operating current range: 0 ~ 36mA • Control signal input voltage range: • Low: -0.3V ≤ Vin ≤ 1.5V. • High: 2.3V ≤ Vin ≤ Vss. • Maximum power consumption: 20W (when the temperature T = 75 ℃).
2	JGY-370 12V DC Worm Gear Motor with Encoder	Technical parameters: • Rated Voltage: DC 12V • No load speed: 30 RPM • No Load Current: 35mA • Load torque: 0.55kg.cm • Load Current: 180mA • Stall Current: 1A • Ratio: 37.3 • Stall torque: 2.2kg.cm • Encoder power: 3.3V
3	Arduino Uno R3	Famous Arduino microcontroller, but its Chinese equivalent. Technical parameters: • Microcontroller: ATmega328 • Operating Voltage: 5V • Input Voltage (recommended) 7-12V • Digital I/O Pins 14 (of which 6 provide PWM output) • Flash Memory 32 KB (ATmega328) of which 0.5 KB used by bootloader • SRAM 2 KB (ATmega328) • EEPROM 1 KB (ATmega328) • Clock Speed 16 MHz
4	15V Case	15V battery case for 10 AA batteries
5	BLS DuPont cables	Cables to connect all the parts

Table 1: List of components

2.2 Hardware setup

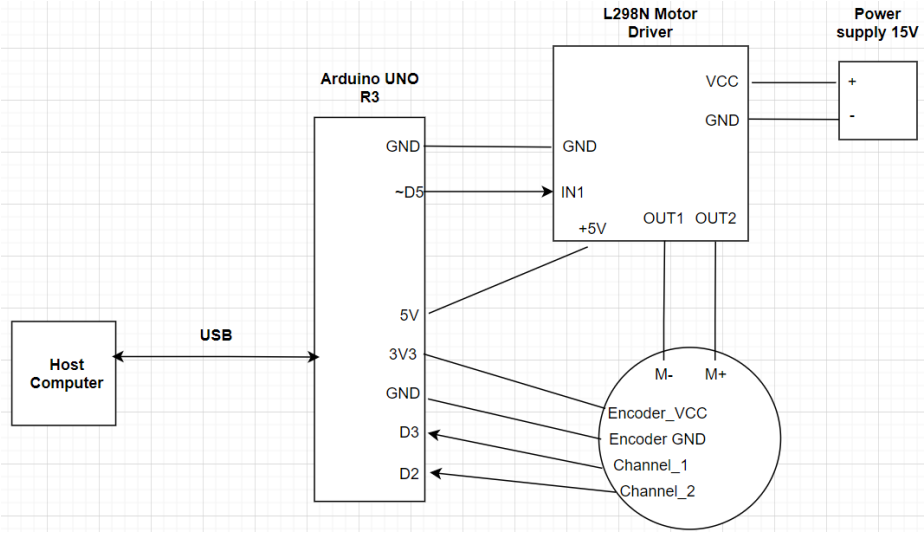

Figure 3: Structure schematic of the prototype

I control the DC motor with the help of L298N driver. As a power supply I use 15V instead of 12V, because, in practice on the output pins of the motor driver voltage is lower than the supply voltage on 2-3 volts. Also, the motor is just 30 RPM, so it's quite slow and I try to get maximum efficiency and speed.

Let's observe the assignment of each line in structure schematic:

- No-arrow lines are supply lines
- Arrow lines from channel 1 and 2 to D3 and D2: lines to send pulses from the encoder when the motor is rotating. They a read via Arduino digital port. Their purpose is to perform a function of feedback.
- Arrow line from ~D5 to IN1: line to send PWM signal to the driver and rotate the motor.
- Double arrow line from Host Computer to Arduino: USB connection between computer and Arduino to exchange information via serial port. Also, a power supply for Arduino.

One more feature. When the supply voltage for L298N is more than 12V, the inner 5V voltage stabilizer, which supplies the logic, can overheat and burn. To avoid this scenario, I take away a jumper on the board and supply the logic separately using Arduino 5V pin.

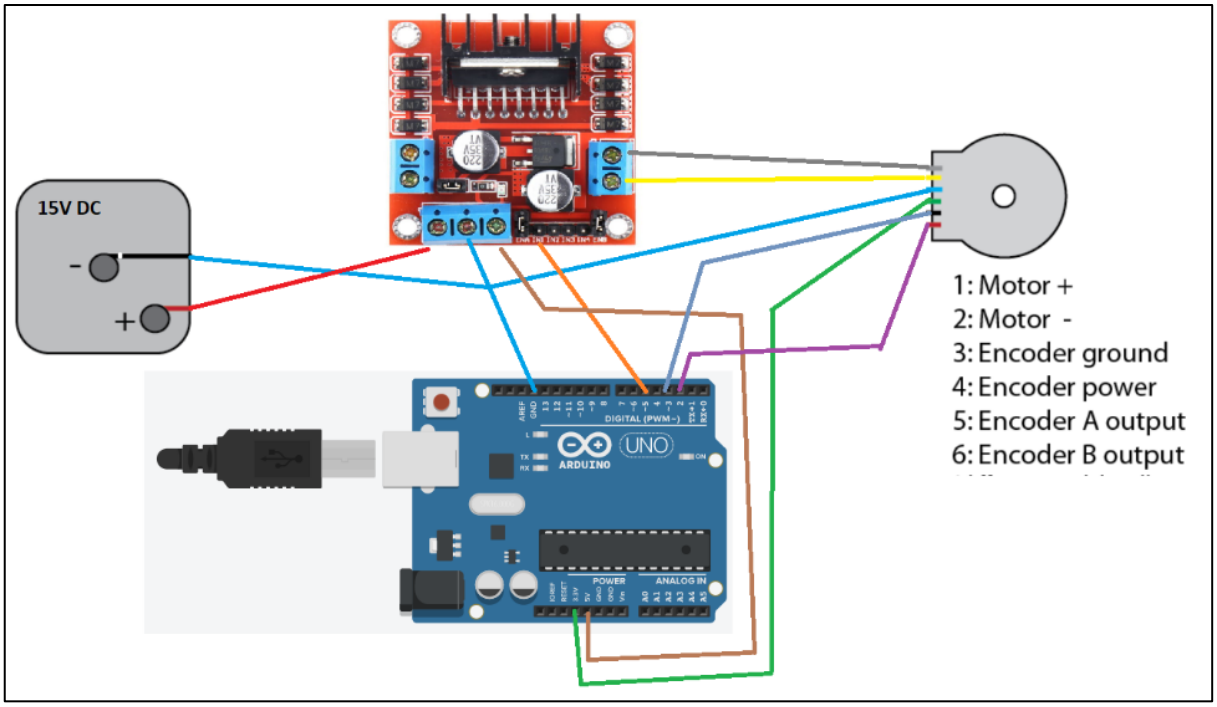

Figure 4: One more connection diagram

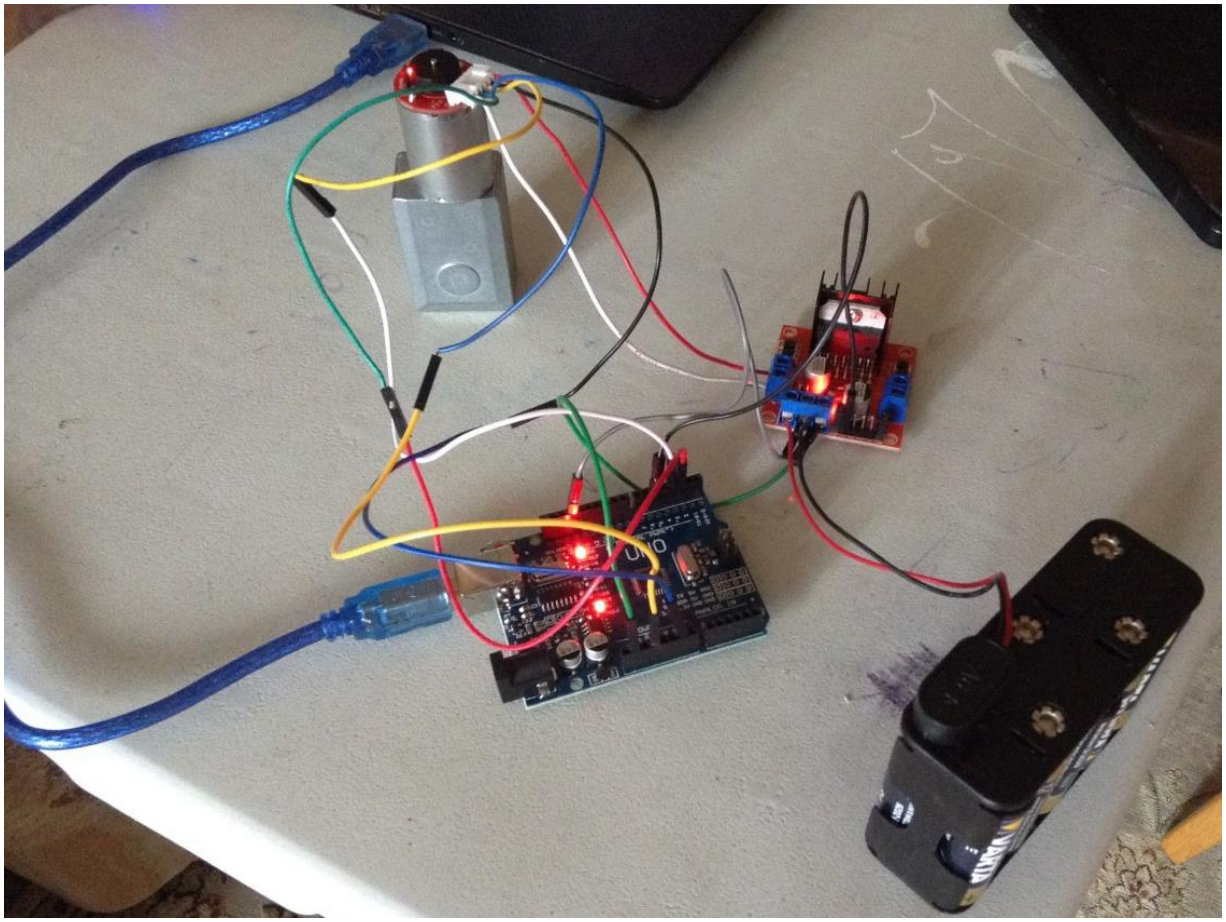

Figure 5: Prototype photo 1

2.4 Software setup

Firstly, I install Arduino IO support package for MATLAB. To make all the blocks work, I need to upload a server program on the Arduino board via Arduino IDE. This program works in parallel with Simulink and will allow to control the board directly from MATLAB. This program uses a serial port to communicate with host computer and executes received commands.

Now work in Simulink begins. Figure 4 shows the first block diagram, that is used to run a motor and read data from encoder. Real-Time Pacer block is used to synchronize time of simulation with real time. I set sample-based pulse to Arduino digital output pin 5 with sample time 0.02 second. Period is 28 seconds and pulse width is 26 seconds. Step magnitude is 1.

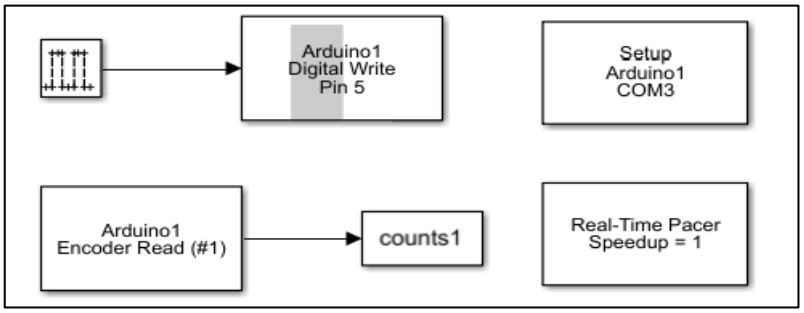

Figure 6: Opening block-diagram to make the motor rotate through Simulink

PART 3

BULDING-UP A MODEL

3.1 Motor run and encoder read

When everything is ready, I can launch the motor. In this section I will obtain the output of the motor in RPM. Then, based on output and input I will be able to determine the transfer function of the motor, and so design a PID controller for it. When I begin the simulation and plot count versus time, I get such a picture (fig. 7):

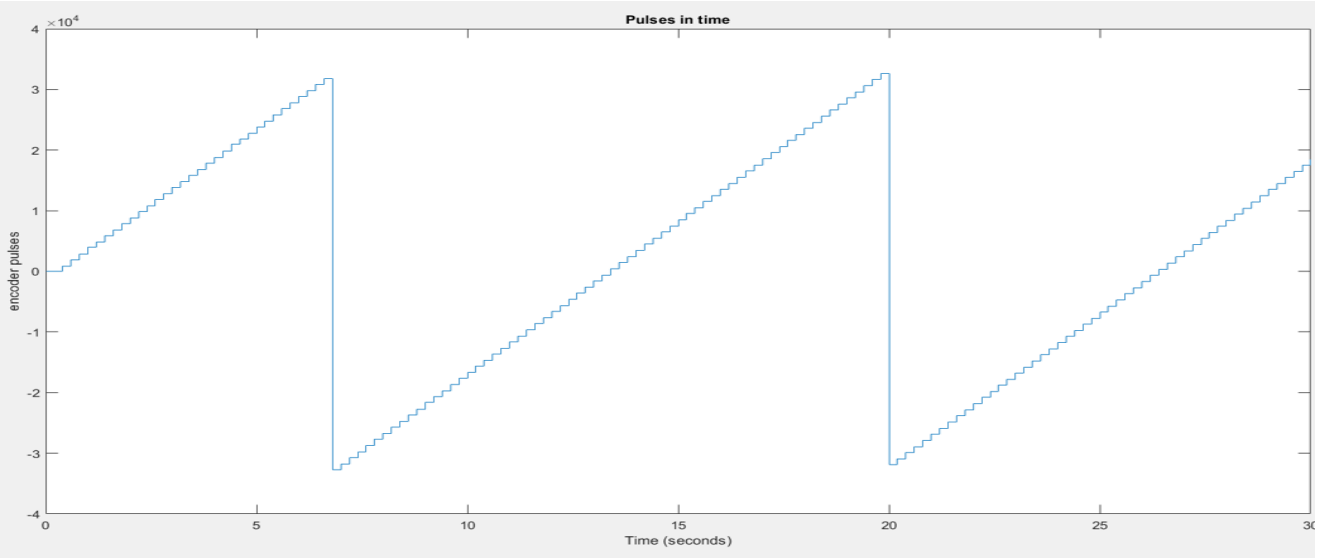

Figure 7: Buffer overflow after receiving pulses from encoder

As we can see, there is an overflow of data, because the buffer in which it is collected is just 16 bit (1 for sign and 15 for value), so I implemented a function to get rid of the overflow. Its block diagram is shown in figure 8.

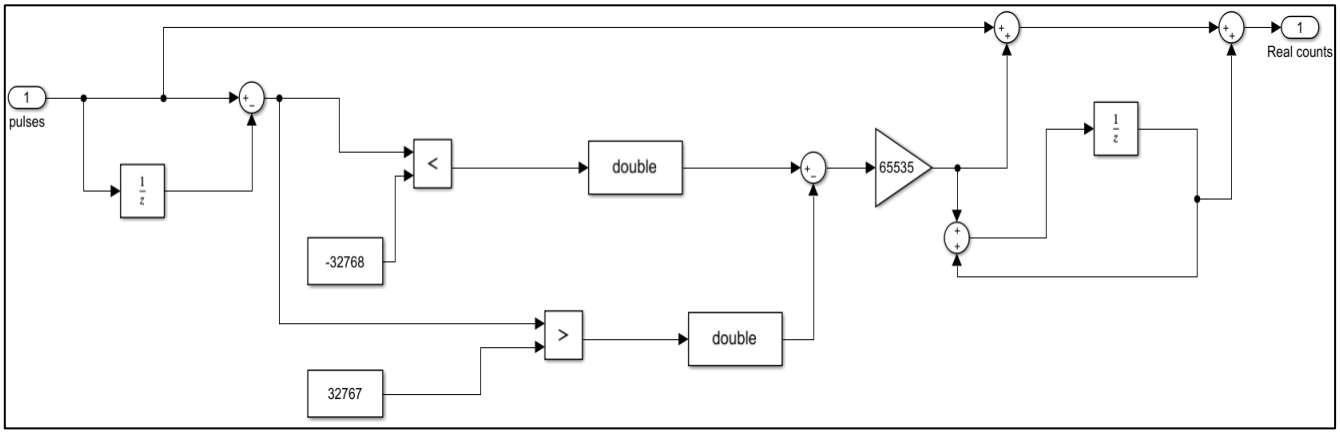

Figure 8: A function that takes the overflow away to further data processing

I compare counts of present and previous sample, to determine whether the overflows occurred in the -32768 pulse or 32767, when I accumulate the value and continue to count impulses. After this correction I get this block diagram and plot result (figure 9 and 10):

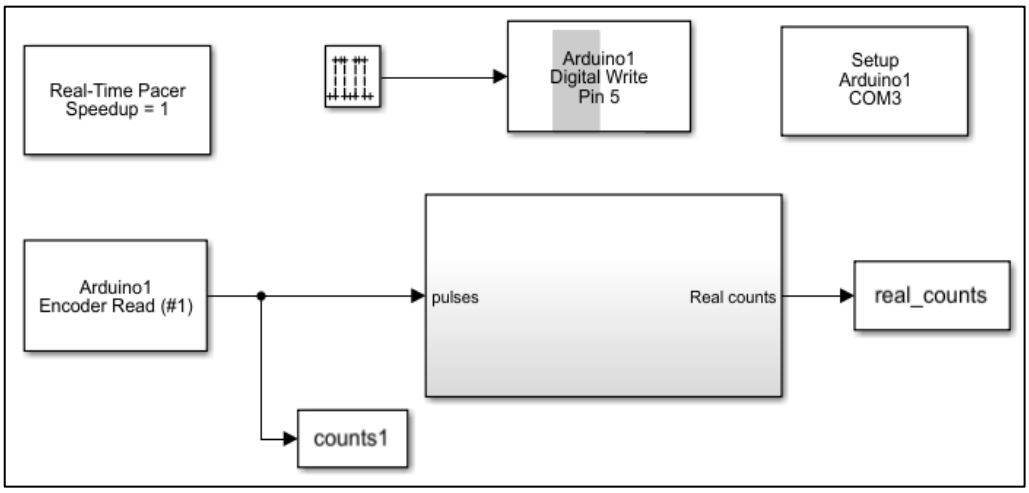

Figure 9: Block diagram after applying anti-overflow function

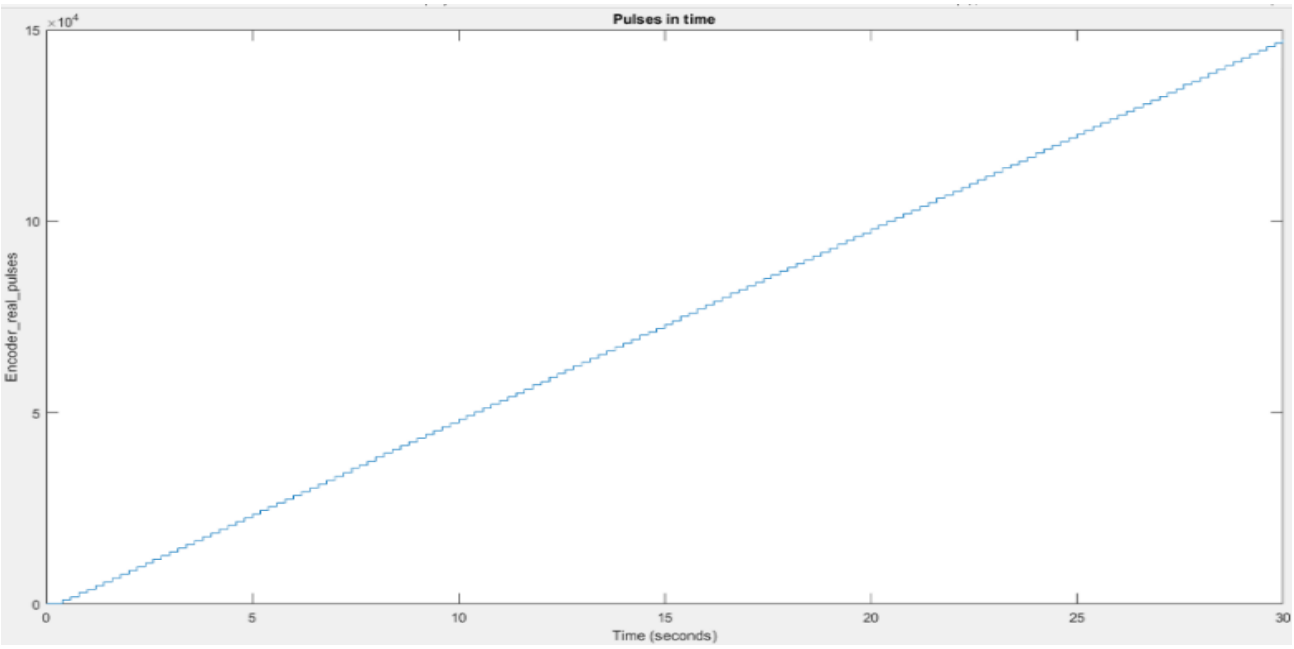

Figure 10: Motor counts without overflow

3.2 Determine the RPM

Now I need to determine motor speed practically and plot it versus time. Firstly, I calculate difference in counts (in position) and then divide by sample time(T), so I get counts/seconds. Secondly, I need to know encoder counts per revolution. I had, recently, calculated pulses during 30 seconds when the motor is constantly rotating. I

got almost 150000 counts. At the same time, I know that, theoretically motor has 30 RPM, so 15 rotations per half a minute. Now it's possible to determine pulses per revolution (PPR):

$$PPR = \frac{150000}{15} = 10000.$$

Now I have PPR and counts/seconds. To determine RPM, I have to take counts/seconds then divide by PPR (I get revolution per second) and multiply by 60 to convert into revolutions/minute. The following block diagram represents all the steps above (fig.11):

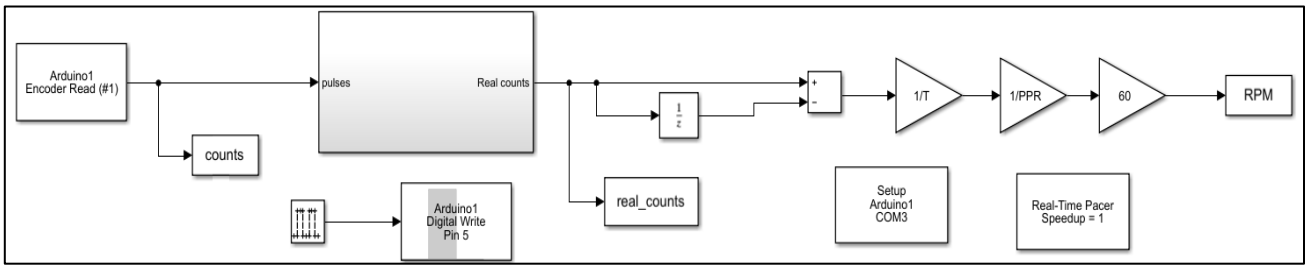

Figure 11: Block diagram with RPM as final output

DC Motor speed plot looks like that (fig. 12):

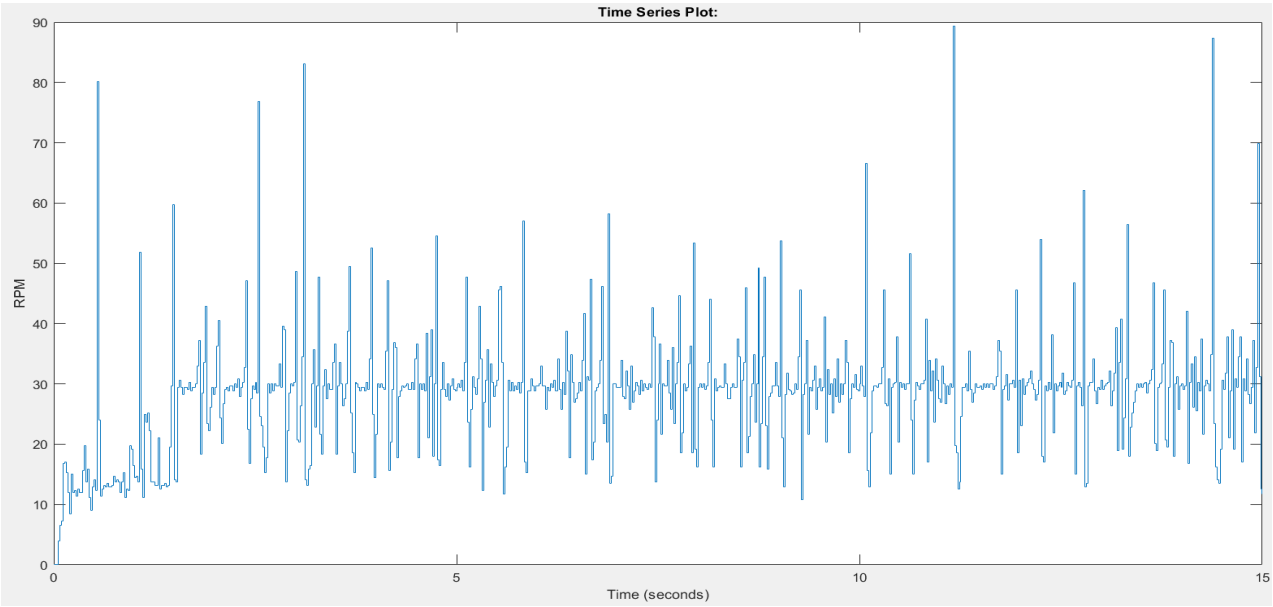

Figure 12: Unfiltered RPM of DC motor

As we can see, the plot looks noisy. It's because of defects in wire connection, so periodically impulses are missed. Also, motor speed isn't accurately 30 RPM, it may vary. In order to reduce this noise, I apply a common low-pass filter that can be represented as this transfer function:

$$\frac{1}{0.8s+1}$$

Block diagram and RPM plot now look like that (fig. 13 and 14):

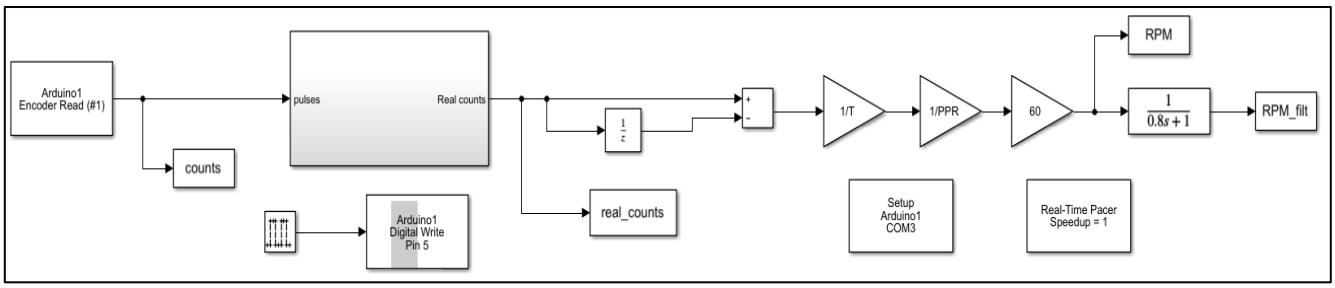

Figure 13: Block diagram of filtered RPM as final output

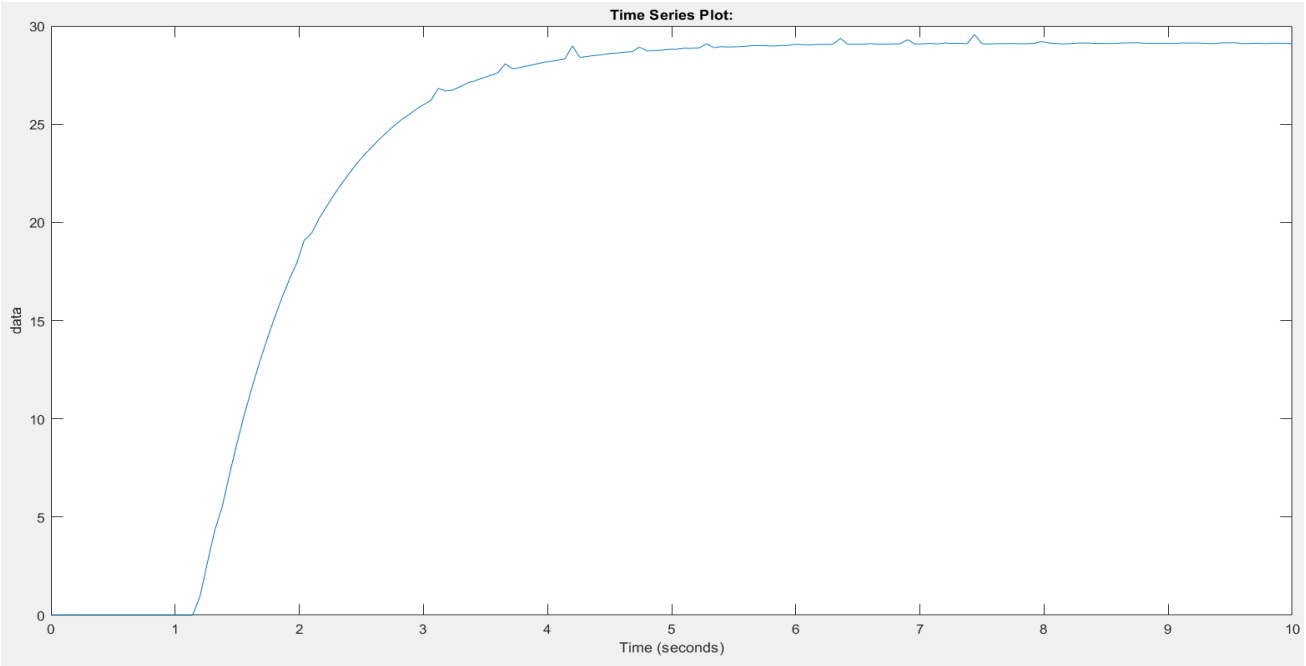

Figure 14: Filtered RPM plot

Now It's much better: plot is quite smooth and we can see that RPM in not exactly 30, approximately 28-29 max. This result was accomplished by increasing sample time to 0.06s and using low-pass filter with 0.8 time constant. Using this time constant gives low response time: speed stabilizes after 5 seconds, but the system becomes more robust. I preferred robustness to fast response with lots of noise.

3.3 Plant model estimation

Now I have the input and the output, so I can determine the transfer function of non-linear model called DC motor. To do this I use MATLAB System Identification Tool.

As I said in Part 1, in practice, when an engineer wants to design a controller for complex plant, it is correct to use the System Identification, because it's a good way to test the model using software, find all the pitfalls of it, because running all the test on real model can damage it, causing material losses. The same thing about the controller. I will build a model of a DC Motor, then build a controller for it, then test it on the software and, in conclusion, run it on the hardware.

As input data I use a step, that has magnitude of 30. Actually, the input is 1, as figure 9 shows, but I take 30 to get coordination in input and output data final value, that is 30.

So, step with 30 magnitude and filtered RPM are sent to the Workspace. To identify the motor model, I have applied next steps:

1. Run System Identification and import time domain data. As input-Step, as output-filtered RPM (figure 15):

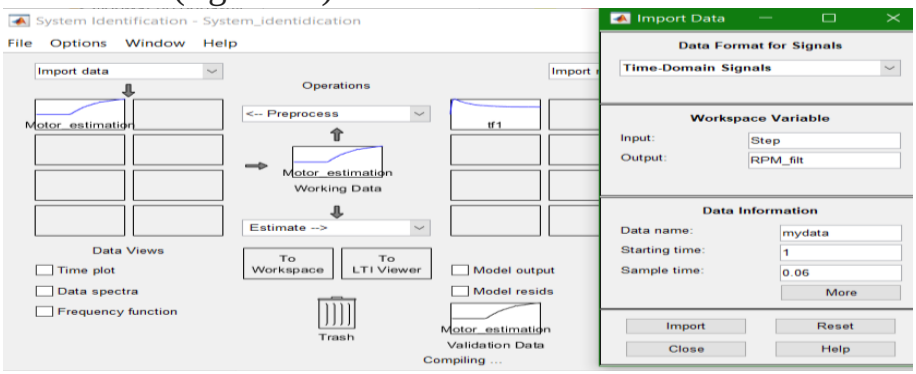

Figure 15: System Identification Tool. Import data.

2. After, I press estimate button and choose a function to be with 2 poles and 1 zero (figure 16):

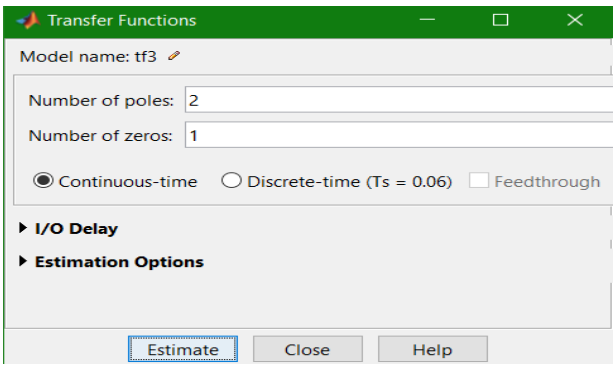

Figure 16: System Identification Tool. Choose function type

3. Run the estimation process and see the result (figure 17). In my case the estimated function matches the real model on 98.24% what is really accurate.

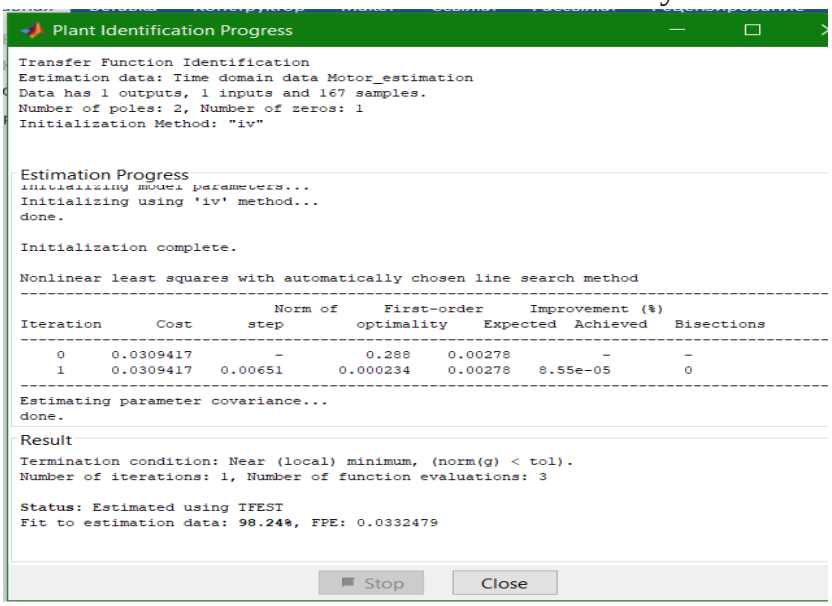

Figure 17: System Identification Tool. Estimation results

After these steps a new variable with default title “tf1”, can be added to Workspace. This is exactly the transfer function I have obtained. In Simulink it can be implemented by adding transfer function block and typing the same coefficients as in the estimated function. In my case:

$$\frac{-0.9717s + 20.12}{s^2 + 17.94s + 20.65}$$

To make sure, I will see how this function responses to step. Figure 18 shows the response result. It’s possible to see, that function’s response is almost the same as real motor speed response, but without noise ramps. I will neglect this feature, because noise is relatively small and will not affect the result considerably. One more addition. To use the data in System Identification tool, it has to be converted into 2D-array. This can be done simply in “To Workspace” block options.

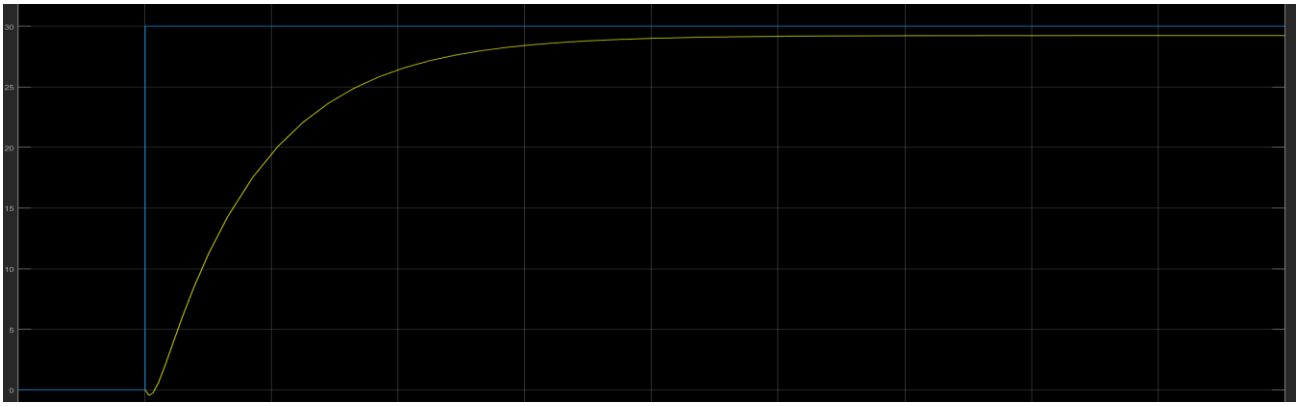

Figure 18: System Identification Tool. Estimation results

PART 4

PID CONTROLLER CONFIGURATION AND PERFOMANCE

4.1 PID controller design

In this part I will design a PID controller for the plant using trial and error method and then MATLAB PID Tuner. The algorithm is next:

- Add Proportional control to reduce the rise time. The larger gain is, the lower rise time.
- Add derivative control and to reduce the overshoot. The larger gain is, the lower overshoot will be
- Add Integral control to reduce steady state error
- Use MATLAB PID Tuner and then compare the two results

So, the first step is to add proportional control. Figure 19 and 20 show the block diagram and function response. I notice, that all of these steps will be done in separate Simulink file, so the motor will not be affected, so while designing a controller we don't test it on the motor, damaging it. That is a power of this approach.

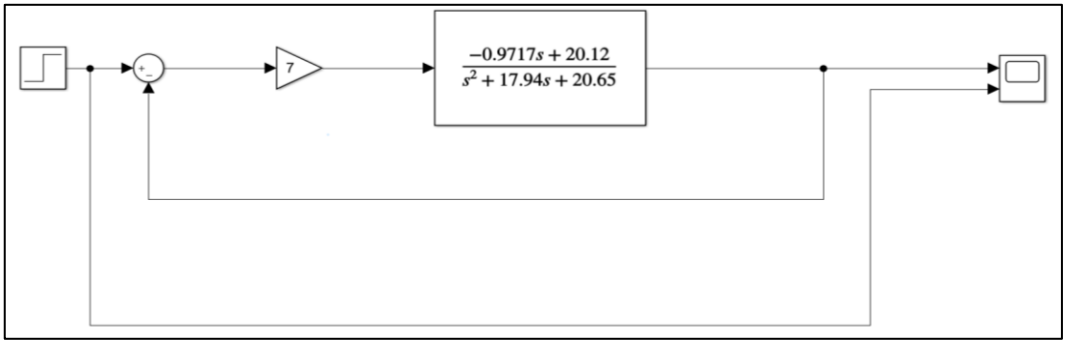

Figure 19: Loop with proportional control only

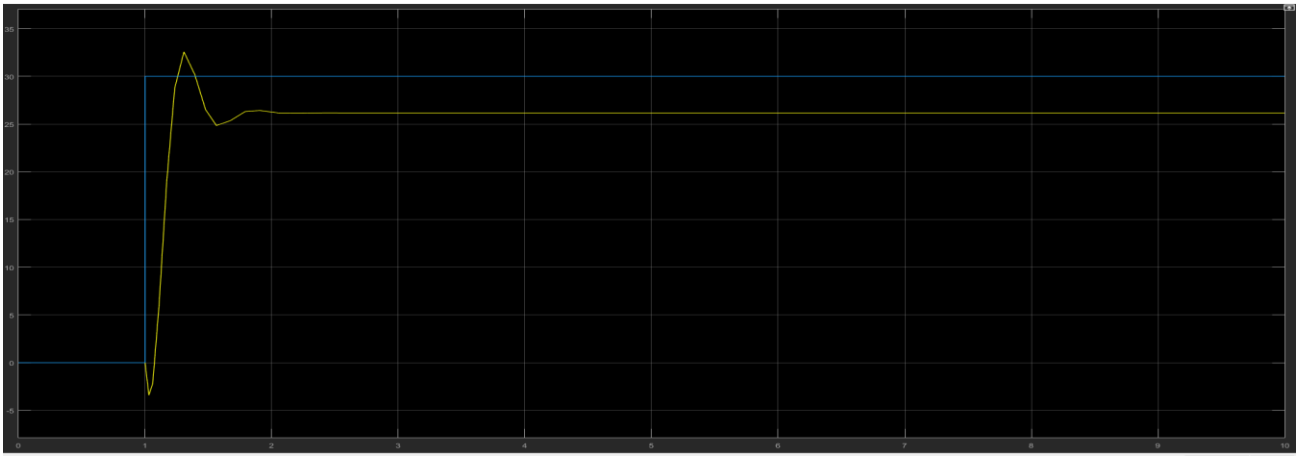

Figure 20: Response with only proportional control

I use gain of 7 magnitude. Applying only proportional part gives fast rise time with some overshoot and great state-steady error. According to this plot, I need some derivative gain, because overshoot is small and relatively large integral gain to reduce considerable steady state error. Next, I apply proportional and derivative blocks. Figure 21 and 22 show the changed block diagram and performance results.

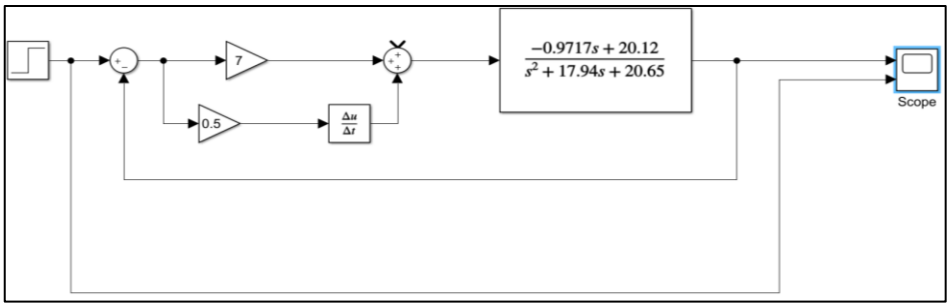

Figure 21: Loop with PD control

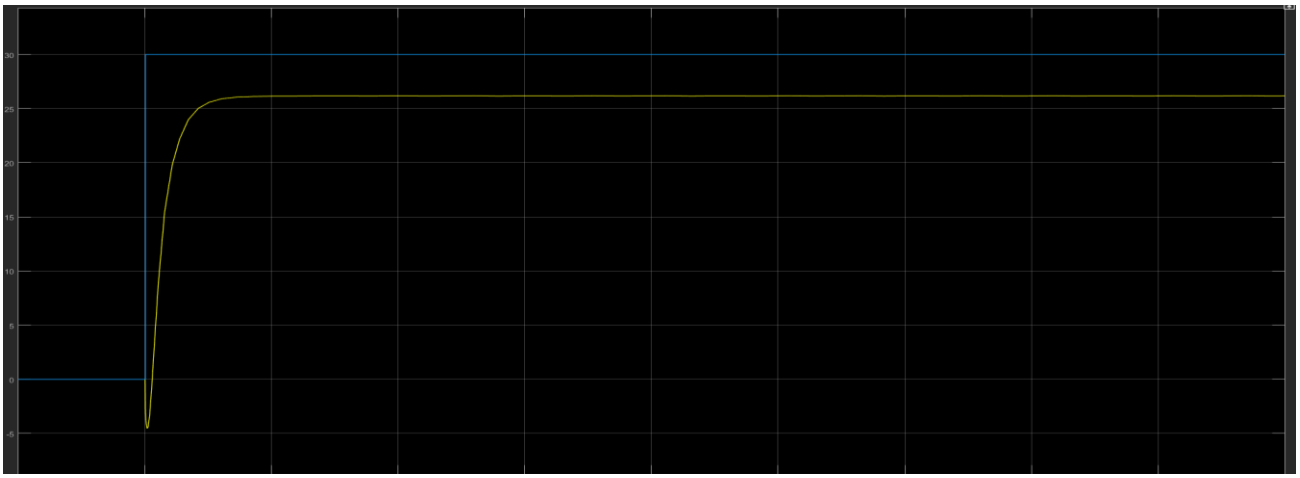

Figure 22: Response with PD control

The last thing is to add Integral part. As I said before, the steady-state error is big, so I need relatively large gain for this part. Let it be between gain and derivative gains, so three. Figures 23 and 24 represent the words above:

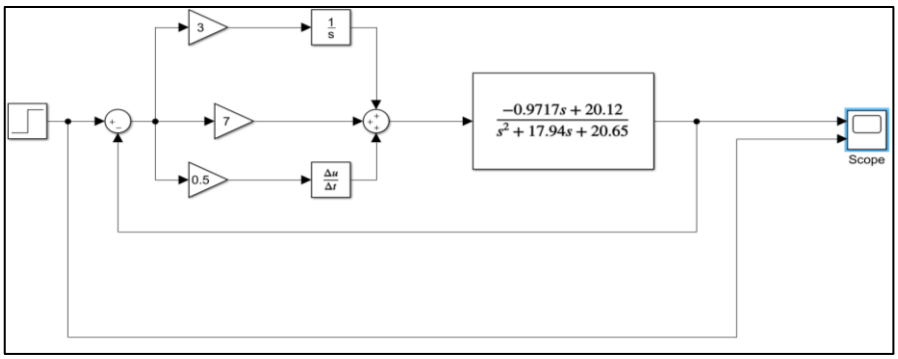

Figure 23: Loop with PID control

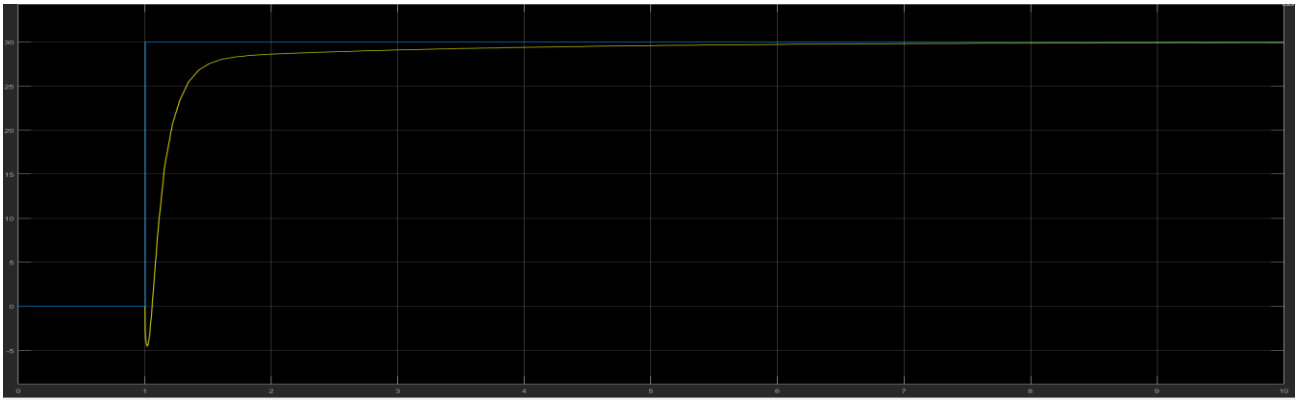

Figure 24: Response with PID control

In this example, that was built with try and error method gave us the result above.

For it I have: Kp=5; Ki=3; Kd=0.5. I made a conclusion, that even without using complex PID mathematics it is possible to design very accurate PID controller. The other issue: how affectively it will work with real system. I will design a second PID controller, but using MATLAB Tuner. The goal is to achieve a little different model response than the previous one and then test each of them on real hardware. Using tuner has given me this result (figure 25) :

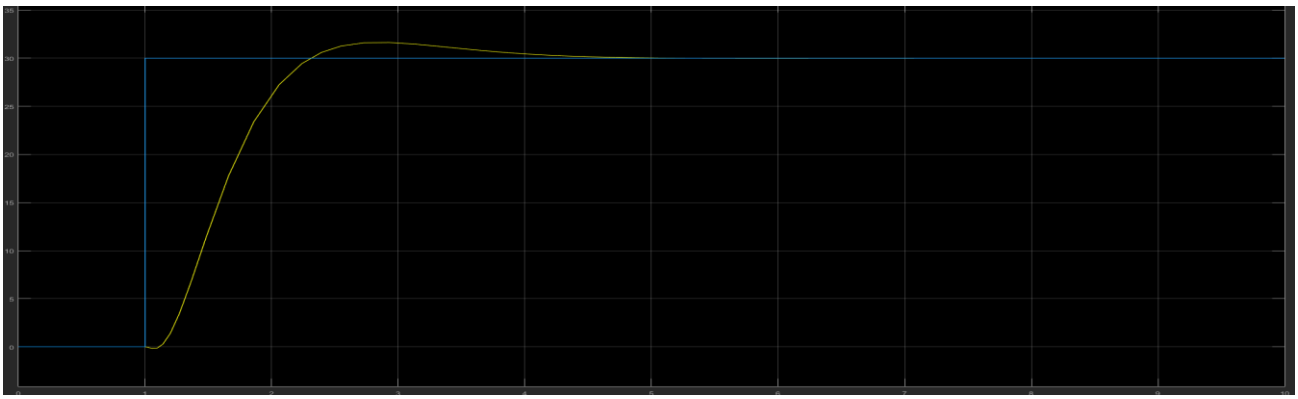

Figure 25: Response with PID controller designed using MATLAB Tuner

Parameters of the controller (table 2):

Controller Parameters

	Tuned	Block
P	0.97517	0.97517
I	1.8738	1.8738
D	-0.11394	-0.11394
N	8.5585	8.5585

Performance and Robustness

	Tuned	Block
Rise time	0.786 seconds	0.786 seconds
Settling time	2.83 seconds	2.83 seconds
Overshoot	5.5 %	5.5 %
Peak	1.06	1.06
Gain margin	17.4 dB @ 7.75 rad/s	17.4 dB @ 7.75 rad/s
Phase margin	63.1 deg @ 1.57 rad/s	63.1 deg @ 1.57 rad/s
Closed-loop stability	Stable	Stable

Table 2: Parameters of the controller designed with MATLAB Tuner

Using MATLAB tool gives more copious information about the gains and the block response, including accurate overshoot, phase margin, settling time etc. Also, I there is a low-pass filter in the coordination with the Derivative part with time constant equals to N. In this example, I have obtained another response with larger rise time and some overshoot, but it's early to make a decision, which of controllers will perform better in real life. Theoretically the first one, but we shall see. Now it's time to test the designed controllers.

4.2 Implementing and testing PID controller on real hardware

In this part I will test my designed PID controllers to control the speed of DC motor. To provide more flexible control of the speed I will use Pulse-Width Modulation of the Arduino 5 port. In MATLAB I have Analog Input block to generate PWM signal, what sounds absurdly, because Arduino cannot generate analog signal, but only this block is responsible for PWM. I use step of magnitude 16, so I want a motor to rotate with 16 RPM. Then, I add sum block and PID controller, so generate appropriate command for the plant. After all, I have to convert a signal to the PWM, so I convert it to bits (from 0 to 255) by adding a 255-gain block. For the first time, I will test controller, implemented by me using try and error method. All words above, represents this block diagram (figure 26) :

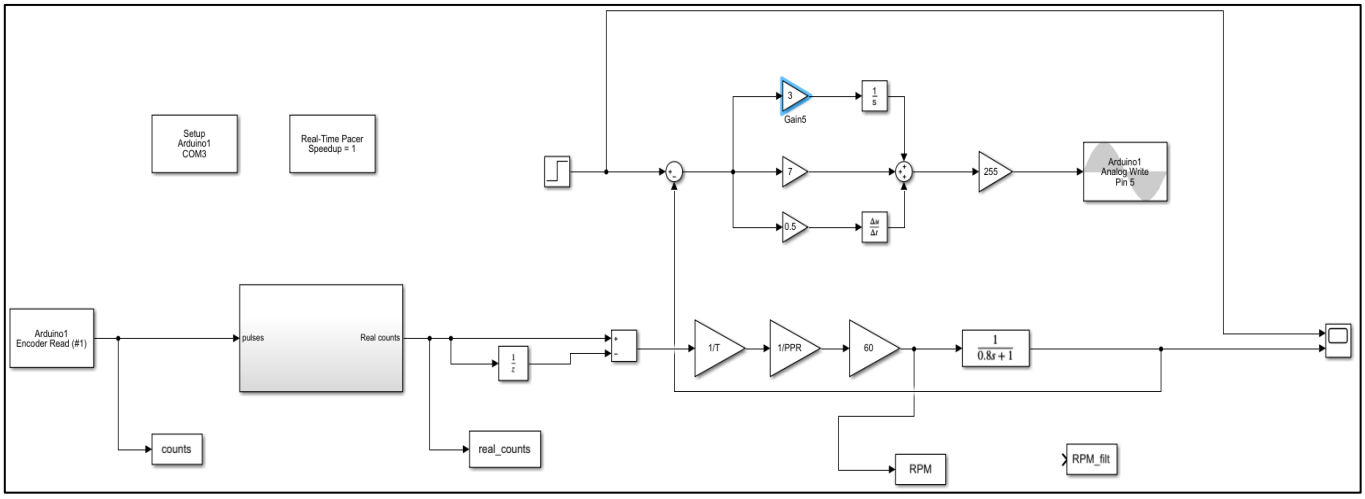

Figure 26: Block-diagram for testing PID controller on real motor

Here is a motor speed response (figure 27):

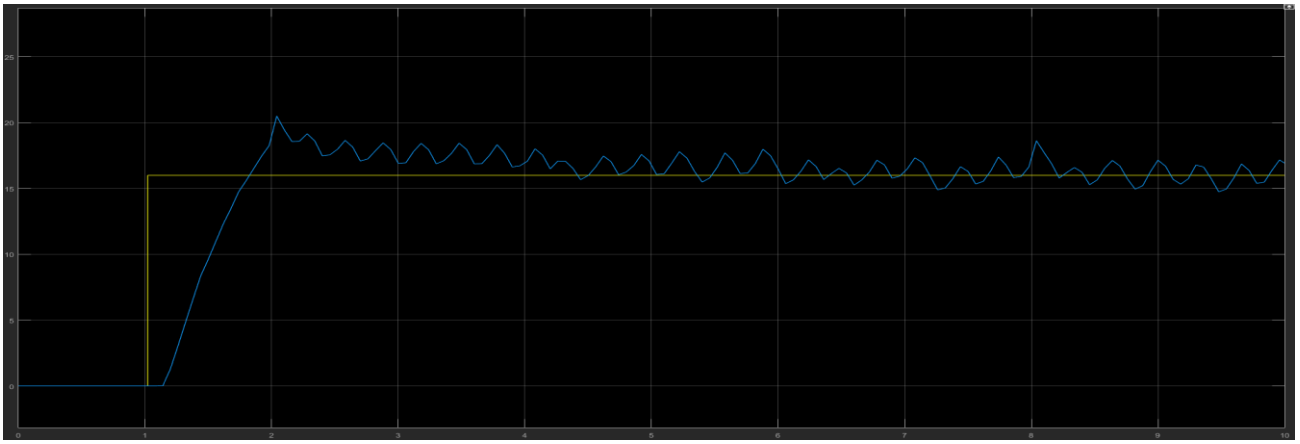

Figure 27: Motor speed response with PID controller with overshoot

Well, the response is not that good. There is a large overshoot and then speed is slowly changing to approximately from 15 to 18 RPM. It's because of the integral windup. It is an occasion, when an integral term is too big and in corresponding with a proportional gain they produce much effort. If I give a command to reach 16 RPM the Integral part begins to accumulate the error very fast and when the integral gain is large plus proportional gain can cause too much effort with great overshoot. To reduce the windup I need to decrease integral gain. I have changed it to 1. Here is a new response (figure 28):

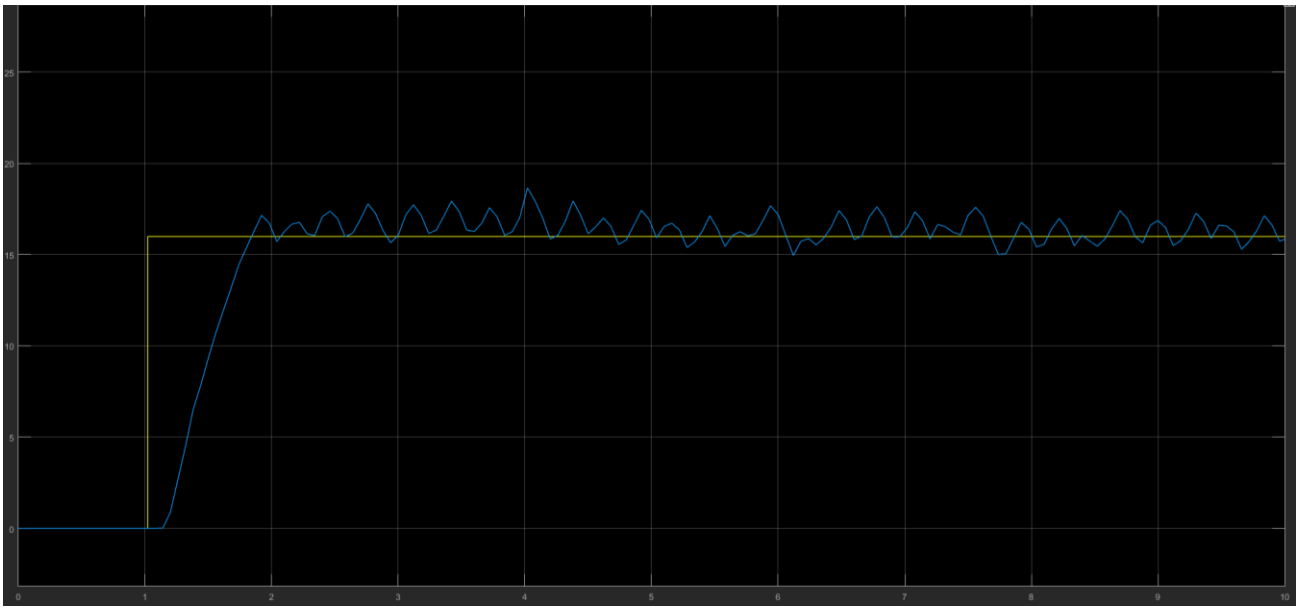

Figure 28: Motor speed response with PID controller

Now it is better. The speed rises fast and then changes between 15 to 18 RPM. I think it is quite accurate result, keeping in mind that wire connection is not ideal at all and there are lots of noises. Now I will test another PID controller, designed with the help of tuner. The block diagram looks like that (figure 29):

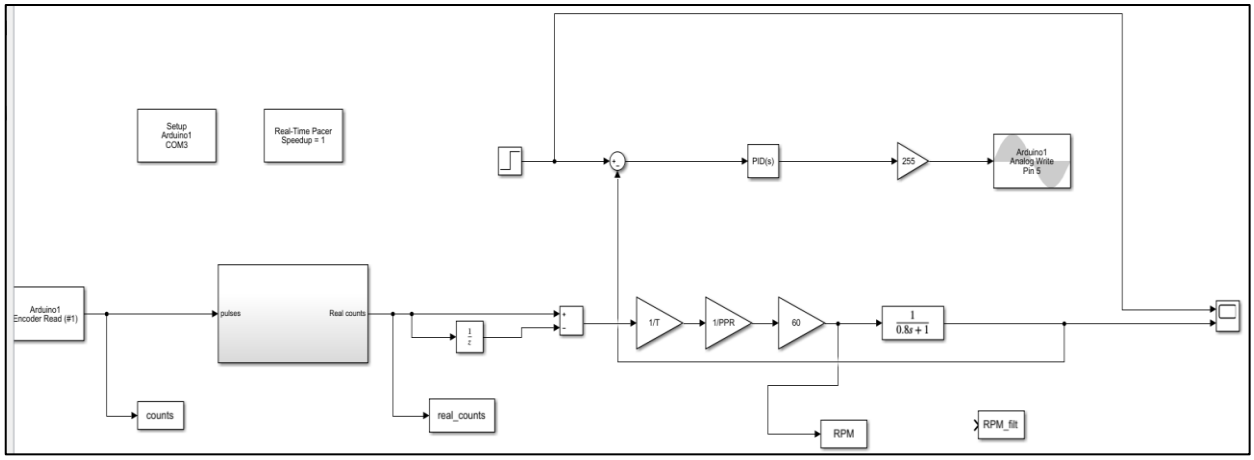

Figure 29: Block diagram with PID controller designed with MATLAB Tuner for testing on real hardware

PID controller above looks more compact now. Here is a response plot (figure 30):

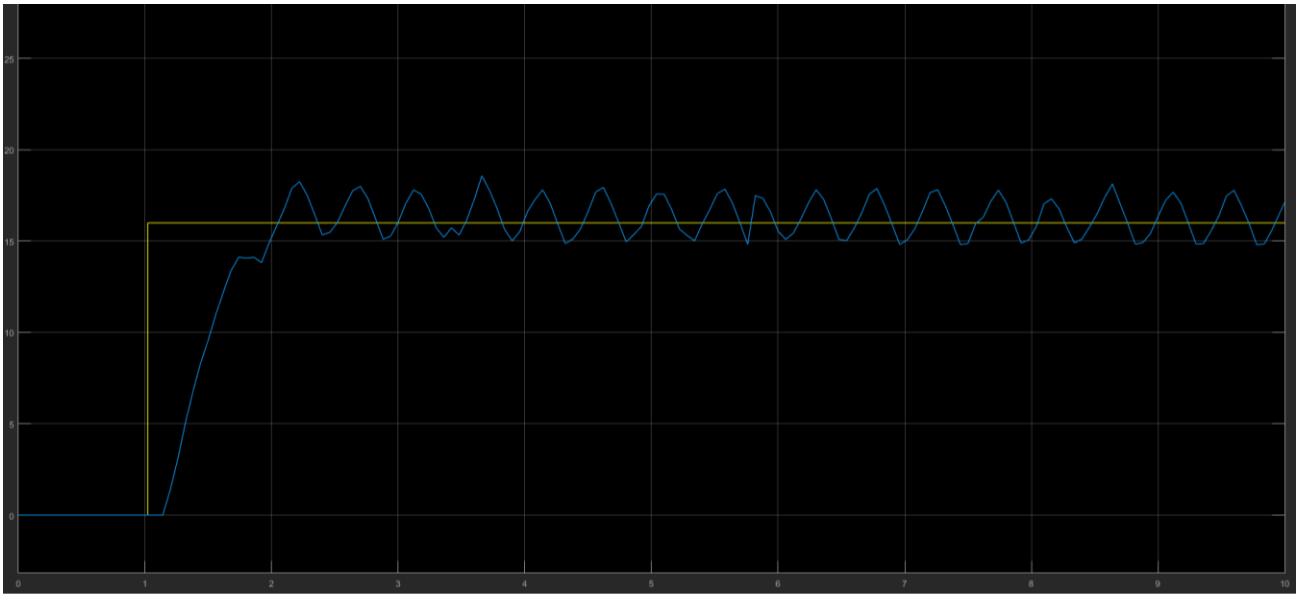

Figure 30: Motor speed response with PID controller designed by MATLAB Tuner

As we can see the result is almost the same as the previous one. The speed is between 15 and 18 RPM. This is the result I wanted to obtain in my course project, because using an equipment I got at home, it is difficult to get more accurate result, but I am sure that it is pretty good, because speed is almost the same as I commanded.

CONCLUSION

In this course project I have designed a PID controller for the DC Motor. For this purpose, I have used MATLAB tools and such an approach allows to builds a controller for a complex non-linear system without knowing its physics. Also, this approach allows to design an accurate controller if the engineer works alone, because the process is almost automized. Controller design topic is very actual, because controller are used in every automized system. Also, I could use a potentiometer to make a controller for the motor position instead of speed. I could implement a PID controller using only Arduino IDE, but making a controller with MATLAB help to get some experience in using MATLAB and SIMULINK. This program is really useful for engineering business. By carrying out my coursework I have obtained basic knowledge in automatic control theory, so it will help in further project where any type of the controllers are used.

REFERENCE LIST

1. Understanding PID control. MATLAB tech talks [Electronic source] — https://www.youtube.com/watch?v=wkfEZmsQqiA&list=PLn8PRpmsu08pQBgjxYFXSsODEF3Jqmm-y

2. PI control design for DC motor. [Electronic source] — http://ctms.engin.umich.edu/CTMS/index.php?aux=Activities_DCmotorB

3. Introduction to control. [Electronic source] — https://www.youtube.com/watch?v=gJzY6jOcgN0&list=PLmK1EnKxphikZ4mmCz2NccSnHZb7v1wV-

4. 298n motor driver datasheet. [Electronic source] — http://www.handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf

5. Simulink tutorial. [Electronic source] — http://ctms.engin.umich.edu/CTMS/index.php?aux=Basics_Simulink

6. Github link with this project - https://github.com/MrGoshak12/Analog_Design/tree/master/Coursework