Development of a platform to Control the Speed and Position of a DC Motor in the Z Plane.

Igor R. Beraldo
Universitat de Lleida
SI I 2018/19

I. Introduction

This paper describes the materials and methods used to implement the controllers in the didactic platform presented by the professor. During the process an interface had to be programmed to enable the user to interact with the systems in an easy way, this interface was programmed using C# on Visual Studio.

The controllers used are the most common ones that are used every day in the industries and in most of the applications. Our goal is to summarize all the steps taken and problems faced to help others to easily understand how the controllers works and be able to reproduce this work with other platforms.

II. Materila List

- Arduino Due;
- Encoder;
- H-bridge MC33926;
- DC motor BS138F;
- 3D printed motor holder with a 360-degree disk attached.

III. Development

Within this section a brief explanation about each component and its function inside the project is given. A step by step about the development of the math and compensator designing is also shown.

A. Interface.

To properly explain the full project, it is easier to start with the interface, because it will be where the user can interfere with the plant itself and alter its controller type, inputs and see the data from the plant, a preview of it can be seen on figure 1.

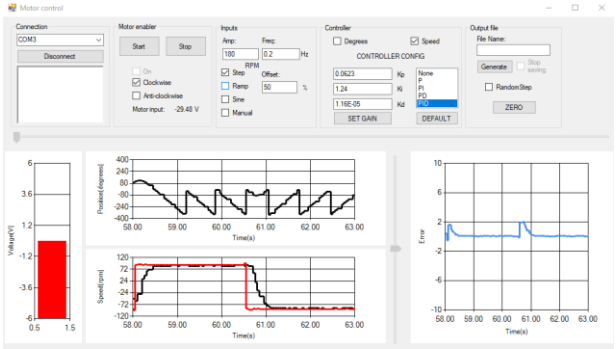

Fig. 1. Final view of the user interface.

This interface was programmed using C# on Visual Studio, the first step needed in order to be able to show data and interact with the plant was to be able to send and receive data from each other (computer – arduino) and the easiest way to do it was through the serial port connection.

The Serial Communication works sending a bit of data at a time in a sequence, for example if the user wants to send the number 10 for the plant using the serial port it will be converted to binary '1010' and the first bit will be sent '1' and the information will be rewritten for the next one '0' after a period of time, which depends of the velocity set until the information is over. That is why it is important to be sure that both devices are set on the same speed for the proper communication. It is also good to know that there is only one way for the information, which means that while one is sending the other cannot send data as well, it must be listening.

With the communication settle it is time to create functions in order to alter the way the plant behaves. As it is supposed to be a didactic platform of control the user will need different sort of inputs to analyze how each compensator effects the plant. It was created 4 kind of inputs, train of pulses, train of ramps, sine and a manual option as well.

To implement any input signal 2 inputs values are required the amplitude of the signal and the frequency, expect the manual one.

- Sine.

With the amplitude and the frequency of the signal saved in a variable it was used on the following eq. 1.

$$Amp\,.\,sine(2\pi ft) \qquad (1)$$

It was not very complex, the frequency multiplied by 2π gives the angular speed of the signal in radian per second, and then multiplying it by time we get radian to get a proper sine value, the result is multiplied by the amplitude that gives the maximum and minimum value of the signal.

- Manual.

This was implemented in a different way, on the C# interface a scrollbar was created that goes from -1 to 1 and it is multiplied by 360 or 110 depending if it is set to speed or position.

- Train of ramps.

This is the second hardest input to implemented, the ramp equation is very simple one and can be seen on the eq 2.

$$ramp = a.t + b \qquad (2)$$

The a is the angular coefficient and the 'b' is the value where the ramp crosses the y axis in this case it is equal to 0 when time is 0. The 'a' can be calculated as the maximum variation of the y axis, which can be 360 or 110 depending if it is for position or speed divided by the time it takes to arrive in the maximum value $\left(T = \frac{1}{f}\right)$.

As a train of ramp is wanted and not a full ramp that keeps increasing forever, it should go back to 0 after achieving the amplitude set on the interface, to do it the rest of the division of the ramp for the amplitude is multiplied by the amplitude. This way after achieving the value set it goes back to zero and restarts.

- Train of pulses.

The most difficult input to implement in my perspective it took my while to remember the Fourier's series that can write any signal as a sum of sines and cosines depending of its characteristics.

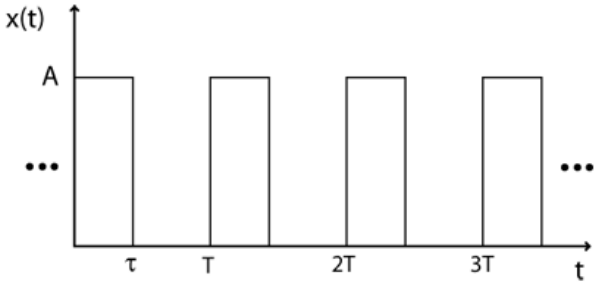

Fig. 2. Train of pulses signal.

As everything before 0 seconds does not exist the mathematical function that describes this signal is an odd Fourier's series it means that the cosines are out of the equation, this way we can describe this signal by the eq 3.

$$\sum_{n=1}^{\infty} \frac{\sin[(2.n-1).w.t]}{(2.n-1)} \qquad (3)$$

The infinite sum means that with more terms more accurate it will be, but a satisfactory result was achieved with a variation of n from 1 to 100. The code to implement it was very easy and it can be seen on the Figure 3 (wo = 2πf, seconds = time).

```
for (int n = 1; n <= 100; n ++)
{
    senos = senos + (Math.Sin((2*n - 1) * wo * seconds)/(2*n - 1));
}
```

Fig. 3. Fourier's series code.

B. Electronics.

Besides the interface a microcontroller is needed to implement all the features desired to this project, and the one that was chosen is the Arduino Due.

The arduino is where all the codes to control the motor will be, basically it is the most important part of the project, it is where the controls will be programmed and it will also be used to read the data from the motor, so basically without a microcontroller, it would be impossible to finish up the whole project as it will lack of most of the features that are needed. However as most of the microcontrollers the Arduino Due does not have much power supply to offer. this is way an actuator is needed in this case the H-bridge is used.

The H-bridge is called this way because of the typical representation of such circuit. The most used diagram to explain it can be seen on Figure 4, this representation gives the proper idea of a H in the circuit that are composed by 4 switches and depending of the combination of the switches the motor spin in a different direction.

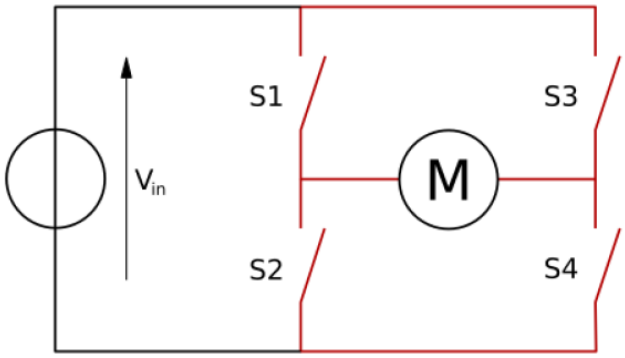

Fig. 4. Graphical representation of the H-bridge.

The one that is being used (MC33926) is composed of MOSFET transistors instead of switches this way the speed of the motor can also be control. This board makes easy to control any DC motor from 5-28V and 3A for each motor, normally each board of these can control 2 motors.

With only 3 cables from the microcontroller, it is possible to control the direction of the motor's rotation and its speed and to break it using the H-bridge which is very helpful, and to understand the signals sent to the actuator it is necessary to understand better the microcontroller used in this case the Arduino Due.

The arduino Due is not a simple microcontroller it is a development board that is ready to use, it is develop on the Atmel SAM3X8E, which is an ARM cortex CPU core and different of most of the Arduino boards this one is 32 bit, it means it can process more information at the same time, so it is faster than the other boards. It counts with the same number of digital and analog ports as the Arduino Mega, that is the most pins in any board. It also counts with the possibility to use more than one timer and interruption pin and it is not very common in most other boards. In the next topics the most important features of the board used will be explained.

- Digital pins.

Theses pins can be used as either input or output and they work with logical HIGH (5V or 3.3V) or LOW (0V), they can be used to read values from sensors or send data to another devices, and most of them have a second or even a third feature as PWM, interruption, SPI and others. Two of these pins are used as output to set the direction of the motor and to break it.

- PWM.

PWM means pulse width modulation, and it is a second feature of some digital pins. It basically generates a signal that alters between HIGH and LOW in a predefined frequency, and it is possible to alter its duty cycle, that goes from 0 to 100%, this is the percentage of the signal that will be on HIGH during

a period. This is wisely used to control the power of something as that when the duty cycle is in 0% the output voltage of the pin is 0V and when it is 100% the output voltage is 5V, so any value between can generate an output value that can be easily found by a rule of three. This feature is used to control the speed of the motor as when the nominal voltage of it is applied it gets to its full speed and while it downgrades the speed also gets lower.

- Interruptions.

These are also another feature of digital pins and what they basically do is to whenever they change, rise or fall depending of the setting in the code, they stop the code and force the microcontroller to go to a specific function and after it they return to the same instruction of the code that it was before the interruption. This feature is used to read the data from the encoder, which is the sensor of the motor.

- Timers.

The timer differently of the others features explained until now is a kind of an intern function, in a way it is like an interruption, the difference is that it does not depend of a pin, but a predefined time to call the function and after doing the instruction of its function it returns to point where it stopped. This feature is used to calculate the speed and position of the motor based on the data from the encoder that will be explained later, and to send the data read to the user interface.

- 12 bits AD/DA.

This is a hardware difference between this board and the others from Arduino, normally the Arduino boards have a 8 bits AD/DA, it means this one can deal with more data at the same time and gives a bigger range in the PWM for example as the 8 bits AD/DA can only goes to a variation of 0 to 255 this one can goes from 0 to 4095 and gives a smother variation of the signal.

Those topics above are the most important information to know about the microcontroller that is in use, knowing how those work the data from the encoder can be handled. However, to better understand the data and how they were handled it is required to know the motor used.

C. Motor.

The DC model used is a BS138F-2S-6-21 from Micromotors S.R.L, in the table 1 is possible to get all data of it.

Typical values at ambient temperature +20% - Tollerance +/-10%											
TYPE			NOMINAL VOLTAGE	L	RATIO TO:1	NOMINAL TORQUE	SPEED		CURRENT	
							NO LOAD	AT NOMINAL TORQUE	NO LOAD	AT NOMINAL TORQUE
			v	mm		Ncm	rpm		mA	
BS138F	6 12	12	6 12	40	12,25	1,5	220	155	<30 <20	100 55
BS138F	6 12	21	6 12	40	21,14	2,5	125	85	<30 <20	100 55
BS138F	6 12	36	6 12	40	35,73	4	73	53	<30 <20	95 50
BS138F	6 12	72	6 12	40	71,54	7	37	28	<30 <20	95 50
BS138F	6 12	149	6 12	40	149,05	14	18	13	<30 <20	95 50
BS138F	6 12	208	6 12	40	208,66	20	13	9	<30 <20	95 50
BS138F	6 12	608	6 12	40	608,61	50	4,3	3,3	<30 <20	90 48
BS138F	6 12	1470	6 12	40	1470,82	50	1,8	1,6	<30 <20	58 30

Table 1. Micromotors data table.

The motor used is the second on in the data table and there is all the information needed to use it as the nominal voltage (6V) and the reduction (21.14:1) , besides those data it is also needed to know that the encoder sensor is attached in the rotor, that means the reductions gears does not apply to it.

D. Encoder.

The encoder is composed by a disc of magnets that has 6 poles alternating in positive and negative, this disc is attached to the rotor of the motor, and 2 hall sensors perpendicular to each other. Each of the hall sensors are connected to a pin set as an interruption in the microcontroller, so every time the value of the sensor changes it call a function where a variable that starts at zero is summed 1 if the value is HIGH (positive pole) and subtracts 1 if the value is LOW (negative pole). With this is easy to know whether the motor is spinning in clockwise or clockwise direction and using the data from table 1 and the information known about the encoder it is possible to calculate the position and speed of the motor. The variation of the signals can be seen on the Figure 4.

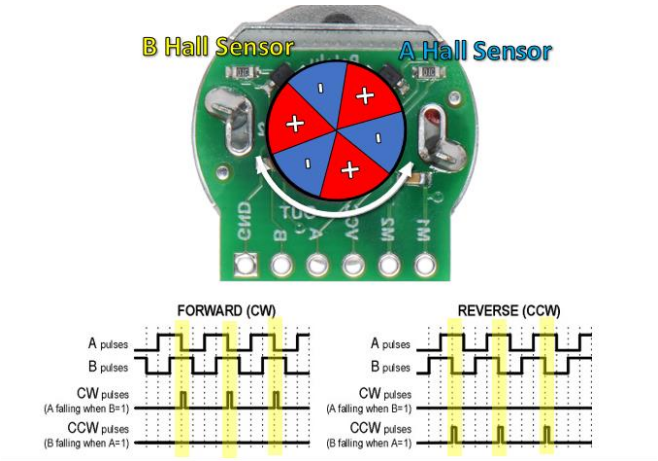

Fig. 4. Encoder view from inside and signals.

In order to properly calculate the speed and position it is needed to know how many pulses the encoder gives in one full rotation of the front rotator as it has a reduction its value must be consider (eq 4), and the relation between a degree per each pulse (eq 5).

$$PPR = 21.14 \,.\, 6 \,.\, 2 = 253.68 \left[\frac{pulses}{rev}\right] \qquad (4)$$

$$\frac{360}{1\,rev} \,.\, \frac{1\,rev}{253.68\,pulses} = 1.419 \left[\frac{deg}{pulse}\right] \qquad (5)$$

Now it is possible to estimate the position of the motor using the constant from equation 5 it is known that each pulse is equivalent a 1.1419 degree, so to calculate it, the following equation is used (eq 6).

$$position[deg] = 1.419 \left[\frac{deg}{pulse}\right] . np\ [pulse] \qquad (6)$$

To estimate the speed can be a little bit trickier as it depends of the time, so now the timer feature of the Arduino is needed to calculate the speed, basically the function called by this timer will check the variation of the pulses (Δnp) and divide it by the time set in the timer (Δt), the result will be the speed in pulses per second, to transform it in RPM the following equation is used (eq 7).

$$speed[RPM] = \frac{\Delta np}{\Delta t}\left[\frac{pulse}{s}\right].\frac{1\,rev}{253.68\,pulses}.\frac{60\,s}{1\,min} \qquad (7)$$

To save time the professor gave a good time to check the variation of the pulses with a small error and, so it will be checked every 0.05 seconds.

E. Plant identification.

In this step it is explained how to obtain the transfer function of the plant to be able to design the controllers. In the interface a function to save the data was implemented, whenever this function is activated it will create .txt file with the data (time, input, position, speed, error, voltage). The transfer function is a relation of the output for the input, so a random step function was generated as an input and the output data (speed) was collected and everything was saved in the file, this file was imported to MATLAB and using its *system identification Toolbox* a second order transfer function without any zeros was assumed for the plant. The result is shown below.

Input	Transfer Function	Fitting
Random Steps (-6V, 6V)	$\frac{130500}{s^2 + 360.6\,s + 7229}$	92,76%

To check if this transfer functions fits the system a plot of time x output was made to compare with the plot of the transfer function's response to the input data that was recorded, and the result is shown in the Figure 5.

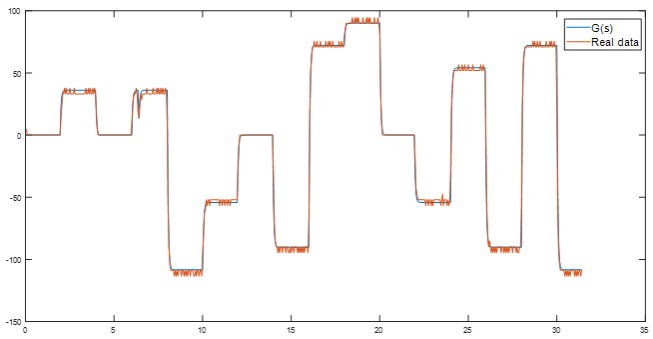

Fig. 5. Plant random steps response and its estimated response G(s).

The graphic shows that the transfer function is accurate and can be used to develop the compensators and a satisfactory result will be accomplished. However, the position plant is different as that a input voltage will have a different effect in the position of the motor, the basic physics says that the position equation is the integral of the speed for the time, but in the frequency world to make an integral of an equation you only need to multiply the equation by $\left(\frac{1}{s}\right)$, as the speed is being calculated in RPM and the position is wanted to be in degree we have to multiplied it by 6.

F. Compensator design.

The designed compensators are the PID and its simpler variations P, PI and PD. As the input data is being sent through the serial port and it is not a continuous signal, but samples of signals it is not possible to work with the frequency domain, So the compensators will be developed in the discrete domain.

The types chosen for the integrative part is trapezoidal and for the derivation part is the backwards Euler, in the following equations is shown how the '**s**' will be replaced.

$$Trapezoidal \qquad s = \frac{2}{T_o}.\frac{z-1}{z+1} \qquad (8)$$

$$Backwards\ Euler \qquad s = \frac{z-1}{z\,.T_o} \qquad (9)$$

The most important part while designing a compensator is to obtain its gains (K_p, K_i and K_d), and as there was not too much time left to test many controls theories, the solution to deliver a good working project was the *MATLAB Application PID Tuner*.

The PID Tuner is an application embedded in MATLAB that tune PID variations of compensators, saving much time and giving a good result. Inside the application one can import the plant of the system one wishes to control and set the type of the controller. As mentioned above the controller will be in the discrete form, so the transfer function of the plant needs to be converted to discrete as well, that can be easily done on the MATLAB by the command ***c2d(G(s), 0.05, 'ZOH')***. In Figure 6 the application can be seen, with the discrete plant imported the type of the controller can be chosen and altering 2 bars a graphic shows the step response of the plant with the compensator implemented.

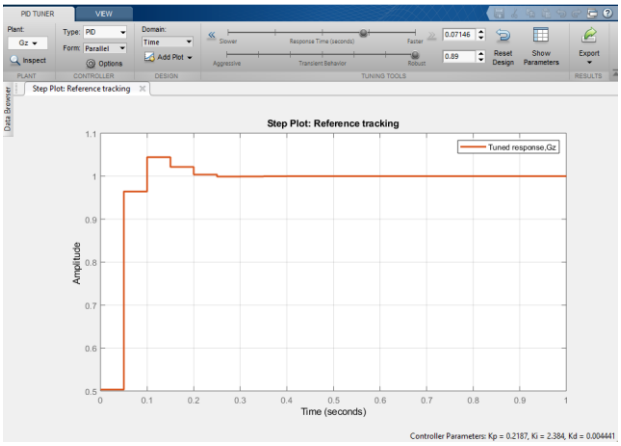

Fig. 6. MATLAB PID Tuner.

In the table 2 and 3 are shown the gains for the four compensators (speed and position) implemented that were obtained through the PID Tuner.

C	*Speed*		
	Kp	*Ki*	*Kd*
P	*0.710*	-	-
PI	*0.012*	*0.497*	-
PD	*0.08*	-	*-1.07e-3*
PID	*0.062*	*1.240*	*1.16e-5*

Table 2. Speed controller's constants.

C	Position		
	Kp	Ki	Kd
P	0.093	-	-
PI	0.088	0.014	-
PD	0.080	-	-0.00107
PID	0.084	0.081	0.003

Table 3. Position controller's constants

With the compensators ready the root locus map of each system was drew to analyze the possible variations of each system. They can be seen on Figure 7 and 8. In all cases the poles are inside the unitary circle, proving the system stability.

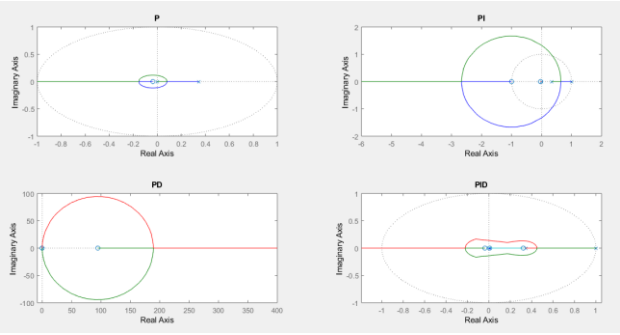

Fig. 7. Speed closed loop zero and pole map representation.

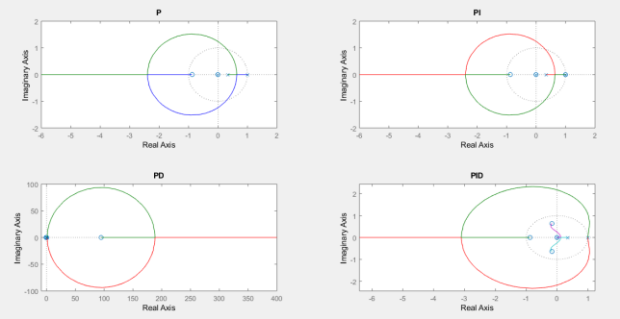

Fig. 8. Position closed loop zero and pole map representation.

To be sure that these compensators will work with all the inputs created, the SIMULINK was used, and the response of each compensator were analyzed to see if they could give a satisfactory response for the ramp and sine, as they were developed using a step as input. And as shown in the results section the compensators are acceptable for all the inputs.

G. *Coding the compensators.*

In order to code a discrete controller (either using a computer or a microprocessor), it must be transformed into its equation in differences. The professor made a step by step example to be follow for all the controllers, and its equation can be found on the following equations.

$$P = Kp.e(k)$$

$$PI = Kp.[e(k) - e(k-1)] + \frac{0.05\,K_i}{2}.[e(k) + e(k-1)] + PI(k-1)$$

$$PD = e(k)\left[Kp + \frac{K_d}{0.05}\right] - \frac{K_d}{0.05}.e(k-1)$$

$$PID = PID(k-1) + \left[Kp + \frac{0.05\,K_i}{2} + \frac{K_d}{0.05}\right].e(k) - \left[Kp - \frac{0.05\,K_i}{2} + \frac{K_d}{0.05}\right].e(k-1) + \frac{K_d}{0.05}.PID(k-2)$$

IV. RESULTS

After coding all the compensators, the data from each one with all the inputs were recorded and with help of MATLAB graphics were plot, where the input and the correspondence output can be seen for each compensator.

- Top: input and output.
- Bottom-right: Control signal.
- Bottom-left: error.

A. *SPEED*

The next figures (9 - 16) show the response fort the step and ramp inputs, of each compensator for speed control developed through this work.

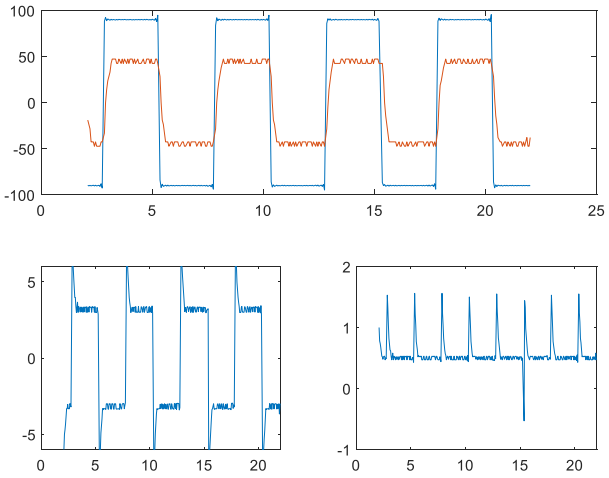

Fig. 9. P Controller response for step input.

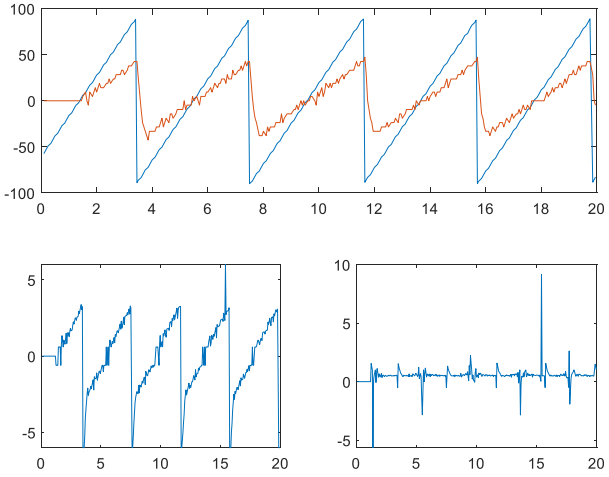

Fig. 10. P Controller response for ramp input

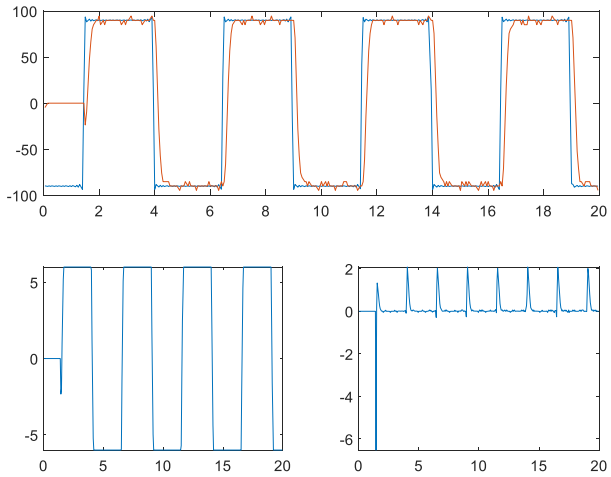

Fig. 11. PI Controller response for step input.

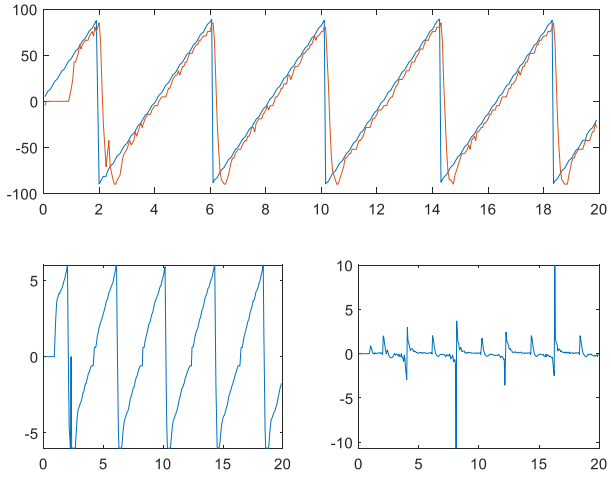

Fig. 12. PI Controller response for ramp input.

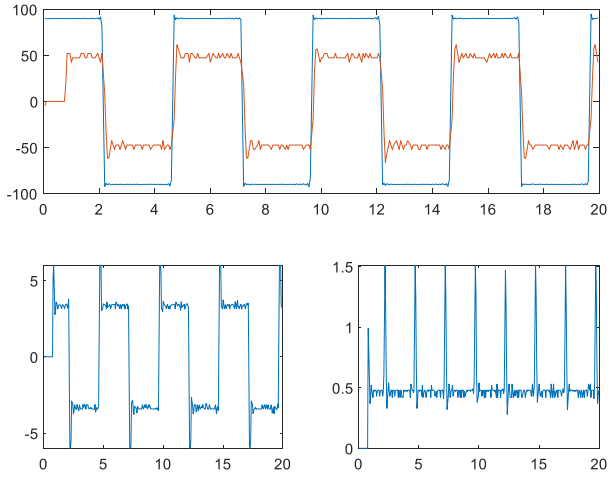

Fig. 13. PD Controller response for step input.

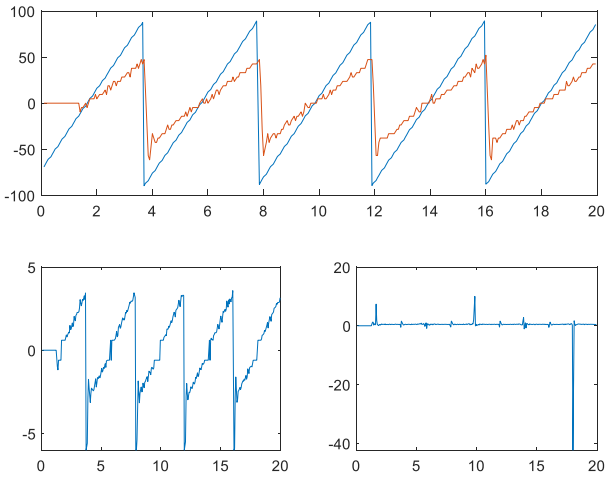

Fig. 14. PD Controller response for ramp input.

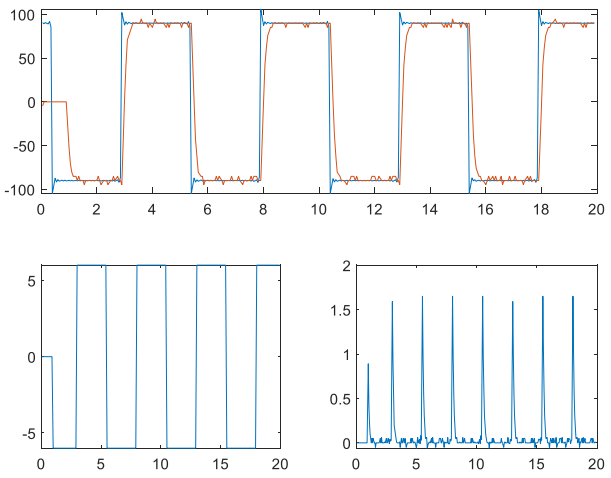

Fig. 15. PID Controller response for step input.

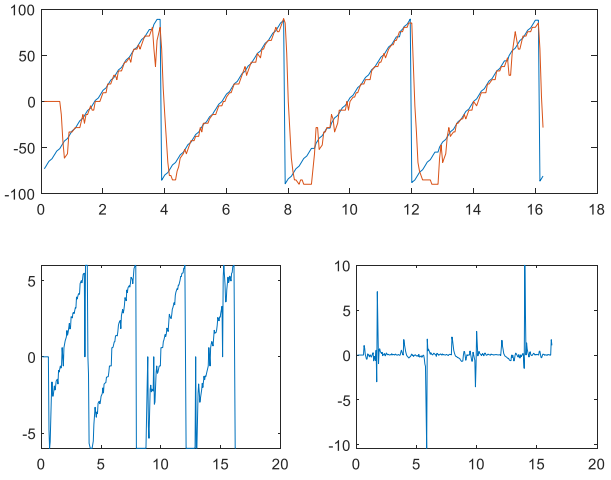

Fig. 16. PID Controller response for ramp input.

B. POSITION

The next figures (17 - 24) show the response fort the step and ramp inputs, of each compensator for position control developed through this work.

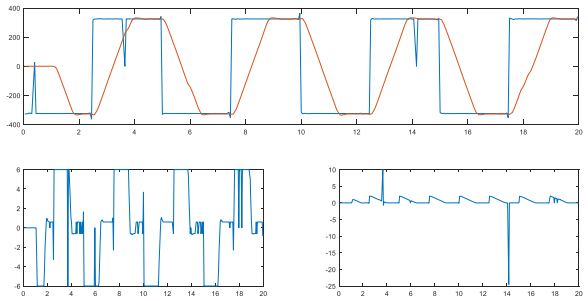

Fig. 17. P Controller response for step input.

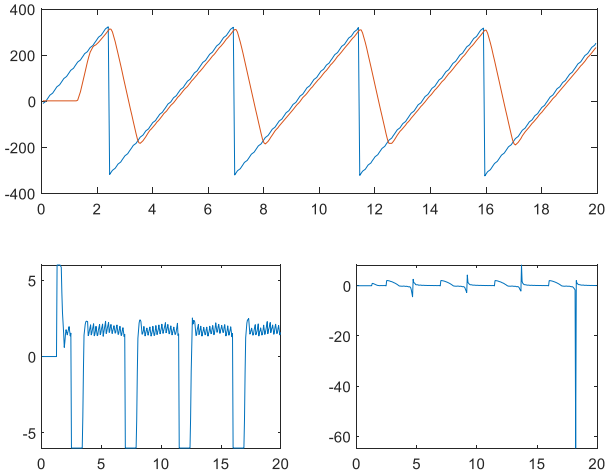

Fig. 18. P Controller response for ramp input.

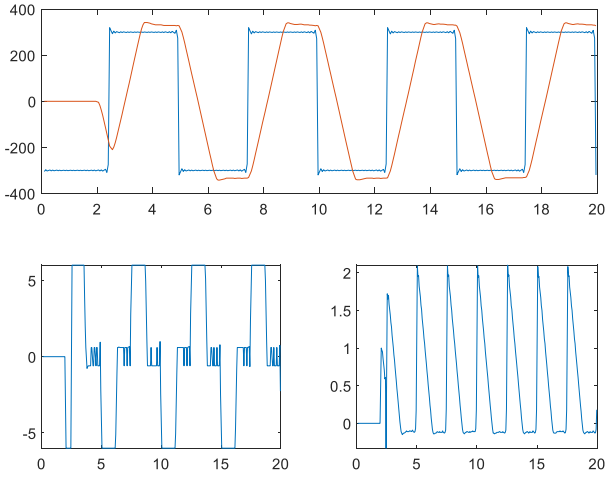

Fig. 19. PI Controller response for step input.

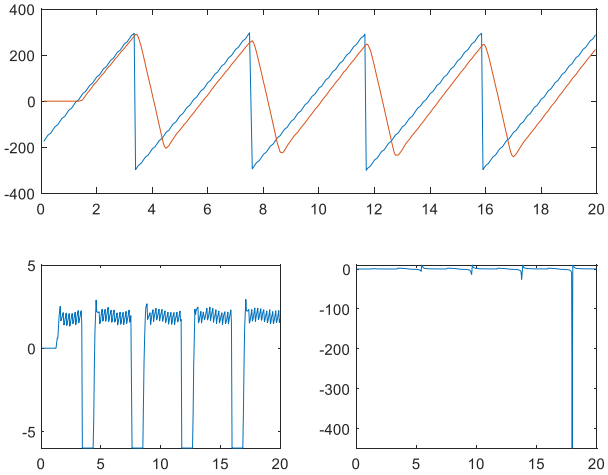

Fig. 20. PI Controller response for ramp input.

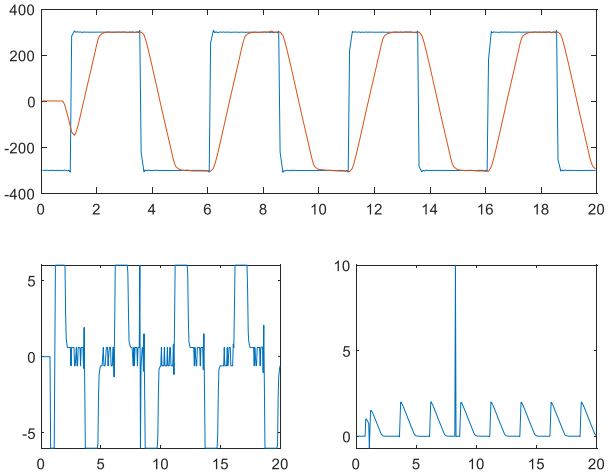

Fig. 21. PD Controller response for step input.

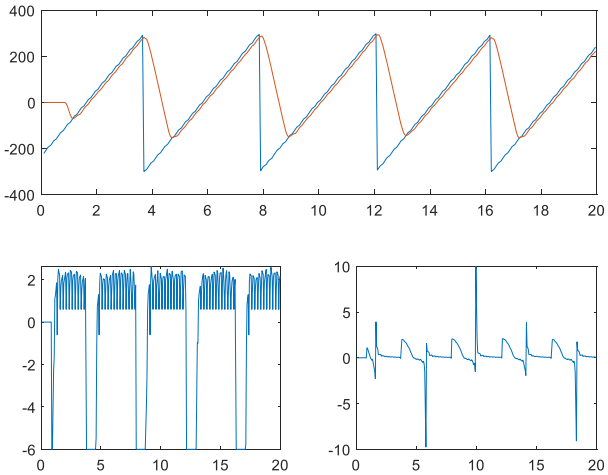

Fig. 22. PD Controller response for ramp input.

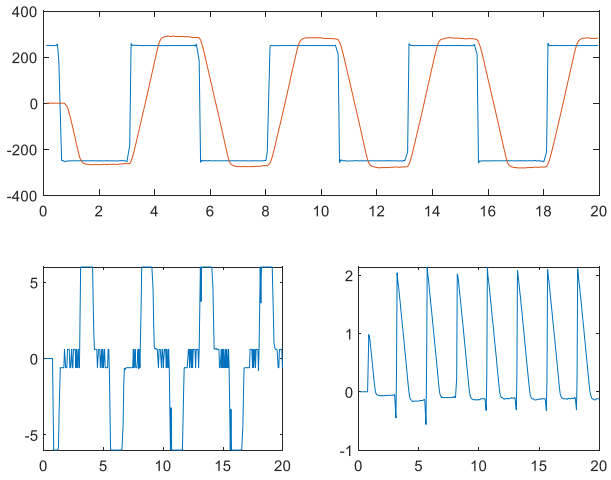

Fig. 23. PID Controller response for step input.

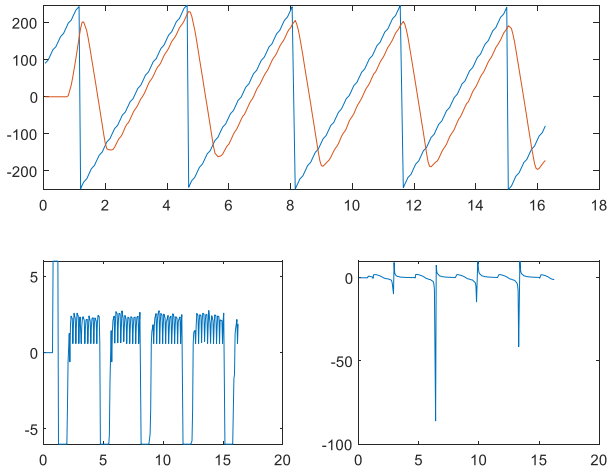

Fig. 24. PID Controller response for ramp input.

V. CONCLUSION

In the figures of the results the inputs are the blue lines and the controllers are the orange one, the speed controllers are all very accurate and they could follow the inputs without any problems. The positions' controllers in other hand had a little delay that I could not fix in time, however they can also follow the inputs without much problem.

During the process of this project I have developed many skills, as it was my first time programming with C# and I was able to do a satisfactory interface and make it communicate with a other device through the serial port, it was also my first time designing discrete controllers and now I am able to them by myself without any problems.