Nick Brannies, Cristian Garcia

PID COntrol visualization

ECE3301 - Final Project

Contents

[Abstract 2](#_Toc24743707)

[Introduction 2](#_Toc24743708)

[Schematic 3](#_Toc24743709)

[Experimental Methodology 4](#_Toc24743710)

[Experimental Results 5](#_Toc24743711)

[Underdamped system 5](#_Toc24743712)

[Critically Damped System 5](#_Toc24743713)

[Challenges Faced 6](#_Toc24743714)

[Hardware 6](#_Toc24743715)

[Designing the Filter: 9](#_Toc24743716)

[Software: 11](#_Toc24743717)

[Conclusion 12](#_Toc24743718)

[References 13](#_Toc24743719)

# Abstract

In this project, we experimented with PID control systems and a way to visualize them in such a way to better understand them. PID systems are widely used throughout the world and this project gives a visual representation of what it is going on in the background to help explain to others or get them interested in the topic of control systems.

# Introduction

To begin, the purpose of this project is to investigate and design a PID control system which will help visualize what a PID control scheme is and what the effects of the PID constants are. Firstly, what exactly is a PID controller? A PID controller is a type of controller which uses a feedback loop which allows the system to target and maintain a value which is generally specified by the user.

For example, someone using a soldering iron may want to set the tip of the soldering iron to by simply turning a potentiometer. The potentiometer outputs a voltage to the control circuit which converts this voltage to a digital value. This digital value is going to have a value determined by how many bits the analog to digital converter (ADC) has. For instance, a 10-bit ADC will convert any analog value to be in a range of:

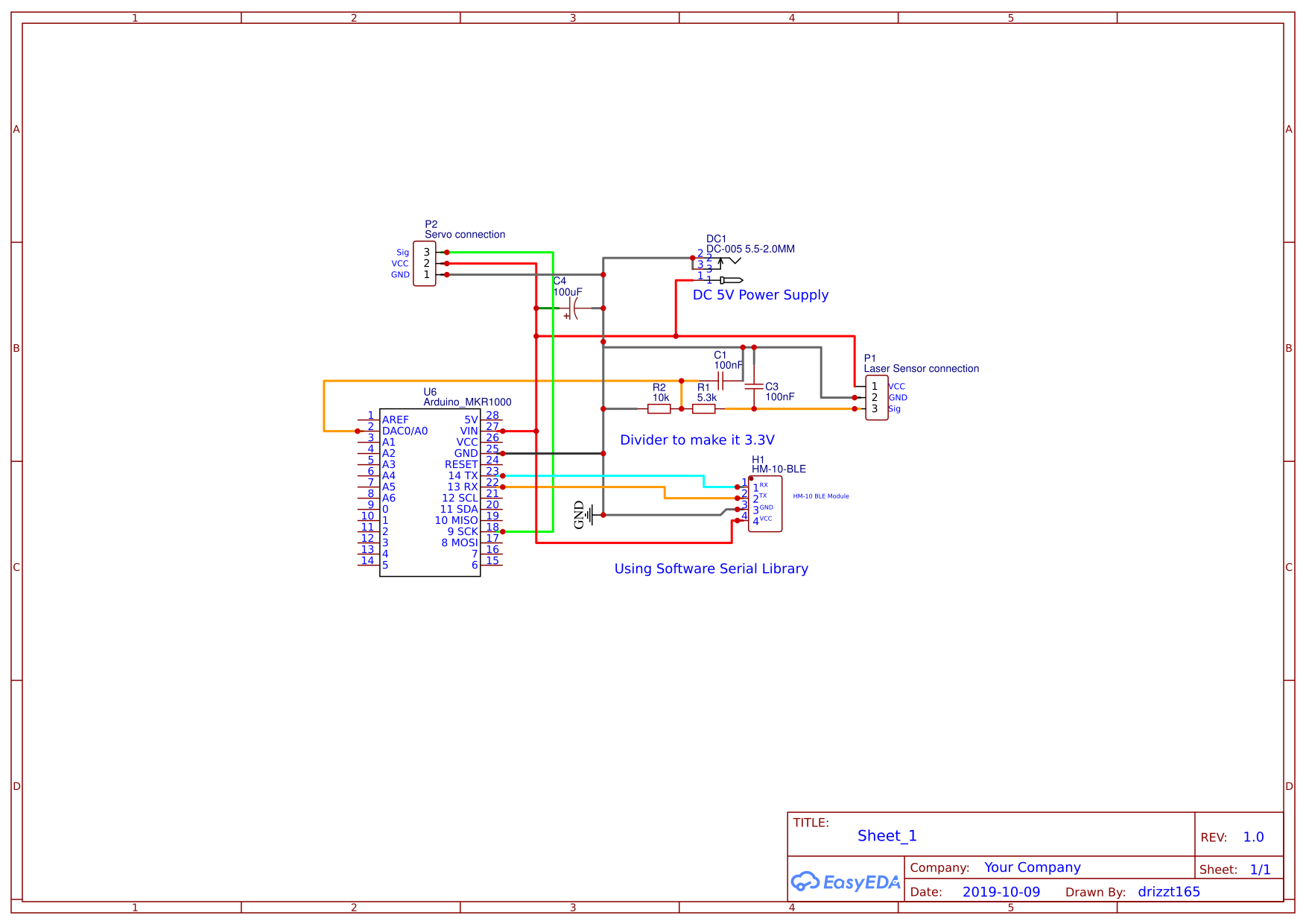
which represents separate discrete values.

Where n is the number of levels or discrete values that can be used. If we expect a voltage value ranging from 0 to 5 volts, then the ADC value ranging from 0 to 1023 must be mapped to represent the value between 0 and 5 or converted to temperature directly. Depending on the values we expect, then we will get a different resolution as follows:

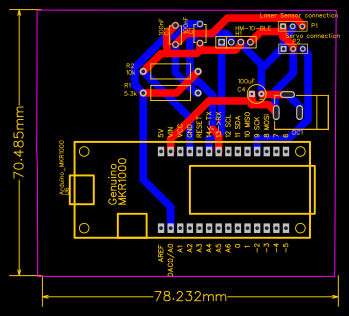
For our microcontroller, the Arduino MKR1000 which operates on 0 – 3.3V, we have a resolution of:

Finally, once the microcontroller has read the input and related the ADC value to a physical temperature value, it can decide on how to change the current to the soldering tip to increase or decrease the temperature. The way this is decided is based of the specific control algorithm decided by the designer and what the target temperature is based on the user input. The algorithm should be designed in a way to reduce the amount of “overshoot” of the target value while having a quick response to change in temperature to maintain the target temperature as well as possible. For instance, the algorithm should increase the temperature a little once the soldering tip contacts solder and cools slightly, but not overshoot.

# Schematic



Circuit - Schematic of project as a whole



Circuit 2 - PCB layout of designed circuit

# Experimental Methodology

The goal of the whole project was to be able to make a system which would balance a ball in the middle (roughly) of a balance beam using a servo and Arduino development board. In addition to this, we added a python script which reads the serial data from the Arduino in order to plot the performance of the system visually with respect to time to better analyze the data graphically.

While experimenting, the PID constants were changed as the system ran which has different effects on the system depending on which constants are changed. The proportional constant is in charge of determining a constant value to turn the servo depending on only the distance the ball is from the target points. The integral constant is for the small changes and fine tuning of the system as it gets closer to the target value. The differential constant is responsible for controlling how fast the system changes or how fast the ball moves in our case. If you think about this enough, you may be reminded of RLC circuits and the equation that is formed when analyzing those circuits:

Where are constants determine from circuit parameters such as resistor, capacitor and inductor values. In a PID system, we have the following:

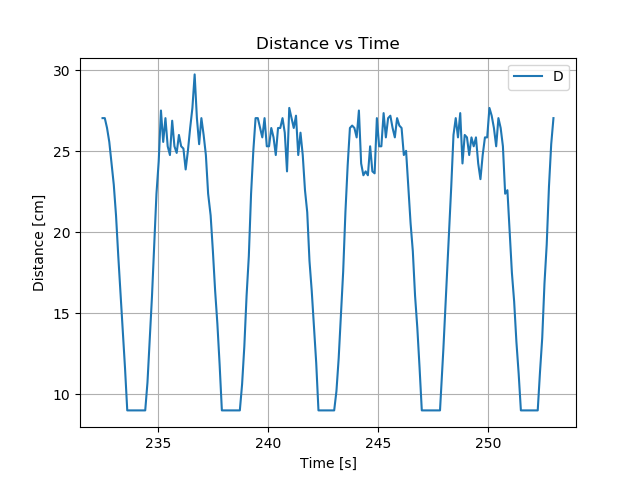
Where D(t) is representing the distance of the ball in the case of our system. Then each component of the system is weighted by their respective constants.

When experimenting throughout the project, we varied the values of these constants and observed the effects they had on the system to develop both the equations above, and to fine tune the constants to have the system behave as we wanted. When doing this we first did the following:

1. Set all constants to zero
2. Varied a single constant to see its contribution
3. Set values of that seemed reasonable from experimentation
4. Varied to get the proper damping
5. Changed to a small number for fine tuning (around 0.1)

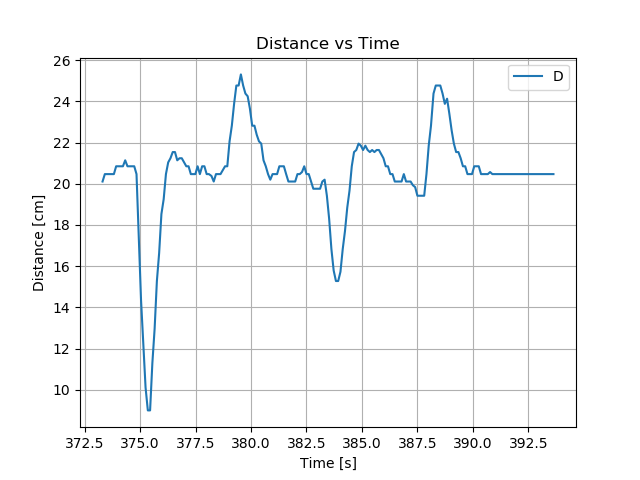
# Experimental Results

## Underdamped system



Plot - Plot for underdamped system

## Critically Damped System



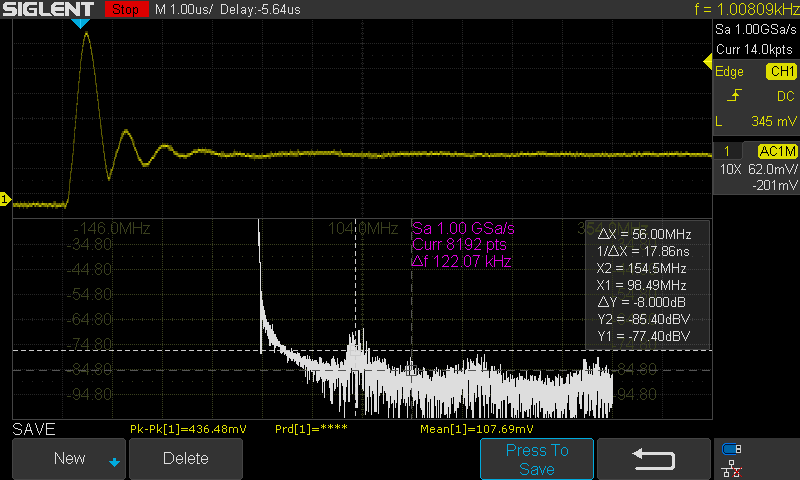
Plot - Plot for critically damped system.

# Challenges Faced

There were many challenges faced and overcome through this project in both hardware and software areas. These challenges include a noisy signal coming from the infrared light distance sensor, communicating with the Arduino via Bluetooth and/or WIFI connection, and designing the circuit with a printed circuit board to get a clean and professional look to the project.

## Hardware

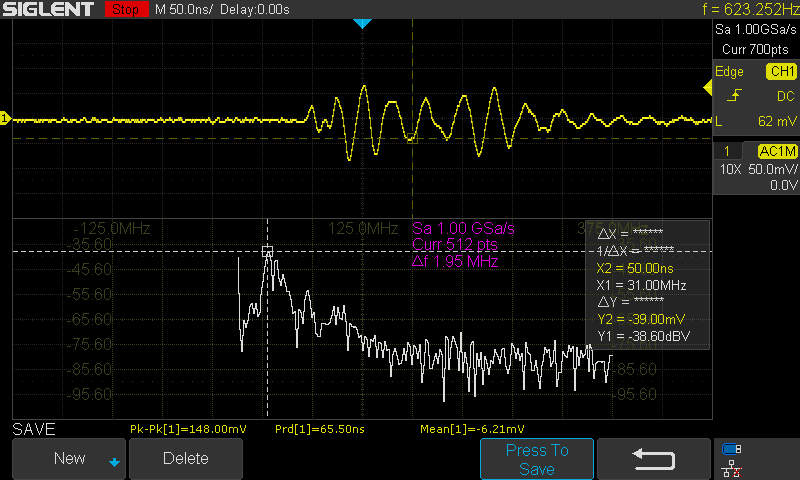
In the hardware side of things, the biggest challenge by far was implementing the infrared light sensor which outputs a voltage based on the distance an object is in front of it. The signal is supposed to be purely direct current (DC); however, we ran into an issue with it having a peak to peak voltage spikes of upwards of 600 -800 millivolts. This caused large amounts of inaccuracies and unstable control where the balancing arm would move wildly in response to this noise. It would still get the ball balanced roughly where the target distance was set; however, it was unpolished and relatively noisy while sitting stationary with so much rapid movement.



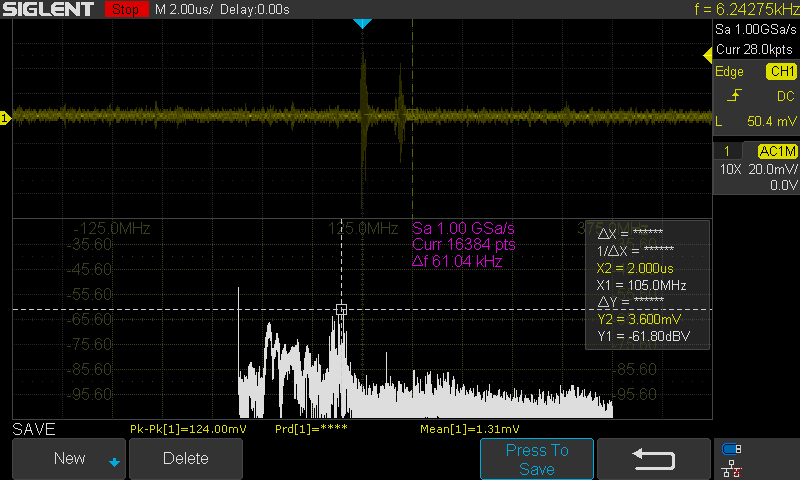
Scope Capture 1 – Scope Capture showing a non-dc component of our signal before filtering

As we can see with the following scope screenshots (using AC coupling mode to remove DC component), there are frequency components well in the 100Mhz range in addition to a fundamental frequency of around 1000 Hz. This causes a peak to peak voltage of 436 mV which is particularly problematic when our signal from the sensor should be a DC value.

Once adding a very simple second order low pass filter, the peak to peak value of the noise was nearly halved. Although there are some higher frequency components still present in the circuit around 30MHz to 100MHz, we were pleased enough at this point for how much better performance the signal was and how it affected the whole project.

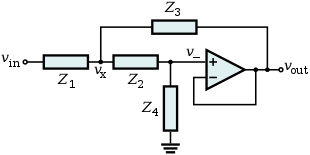


Scope Capture 2 - Screen capture of signal after filtering



Scope Capture 3 - Zoomed out view of noisy signal

With more time, effort, and general knowledge of filtering, a better filter could be built. For instance, there are 4th order Butterworth filters which could be bought online for only a few dollars but require a whole other circuit around it. We could have also used a Sallen key low pass filter as well built from an opamp. However, this would increase the complexity of the project a fair bit.



Since the focus of this project was for on the microcontroller logic and theory, this was thought to be unnecessary for the purposes and goals of the project. Since we were already using voltage divider circuit to step down the 5-volt input from the sensor to the 3.3-volt input of the Arduino MKR1000, we could easily add two capacitors to the circuit, to make a second order lowpass filter as follows.

## Designing the Filter:

First thing to note is that the input impedance of an ideal sensor is ideally infinite, and the output impedance is ideally zero, (it’s essentially an amplifier). Therefore, we can use the little bit of output impedance to help create a second order filter.



Circuit 3 - Filter circuit schematic where V1 is the output signal of the sensor

Circuit 4 - Circuit to find sensor output impedance

In order to get the value of the sensor input impedance, the output of the sensor was set to be a stable value (happened to land on 1.37 volts). Then it was connect to a known resistor in series. The voltage across the known resistor was then measured to solve the sensor impedance as follows.

When selecting values for we wanted to make sure that the values are much greater than the output impedance to reduce the effects caused by it. First, we assumed and calculated the value of as follows:

**Note:**

The resistor value was round up to 5.3k because it is a more commonly found resistor value an readily on hand. Additionally it is within a 5% tolerance of the value.

Plot 3 - Plot of the transfer function for final filter design

Notice that with the first capacitor added to the circuit, we were able to filter out about frequencies past 1MHz almost completely and with the second capacitor, we were able to filter out everything past about 10kHz with a cutoff around 600 Hz. Unfortunately, we saw with the scope, that there were still high frequency components that existed. This is due to capacitors not being exactly like theory. In real life, we would need to select capacitors which are smaller value to filter out higher frequencies as all low pass filters in real life typically have more of a resemblance of a stop band filter as the higher frequencies are still not filtered out.

## Software:

On the software side of the project, there were also multiple challenges which were overcome. One of the biggest challenges was deciding on how we would communicate to the Arduino via a phone and how we would output data to show, graphically, how the system was behaving in real time.

For communicating with a phone, we had a couple of options which consisted of using Wi-Fi and Bluetooth. Initially, we used Wi-Fi due to the built in ESP32 Wi-Fi module which comes preinstalled with the Arduino MKR1000 model. With Wi-Fi in most networks, you only need a SSID which tells the Arduino which network to connect to and a password which is required to connect to the network. The system was up and running quickly with little to no difficulty using Wi-Fi; however, after thinking about it, we realized that the school Wi-Fi (where the demo would take place) needs a username and password which is connected to our school accounts. There is no easy way to connect to the Wi-Fi network with this system in place, so we decided against using Wi-Fi in the final product. Bluetooth was then chosen for communication, however this comes with different challenges of more complex code and using another phone app to connect to the Arduino. After research, and hour of debugging, the system was fixed and running.

Secondly, the other software challenge was to plot the distance of the ball using serial communication in real time. The first decision was deciding on which programming language to use to do this. We thought about using Processing, Python, or even the Arduino serial plotter itself. Immediately though, the Arduino IDE’s serial plotter was removed from this list as it was not very customizable and had many limitations with what we could do with it. Then we had either processing or python to choose from and we settled on python due to it being a hugely popular programming language in industry and it had a module (or library) to easily plot data. Once the python script was up and running, there were apparent issues with it lagging behind the Arduino. On average, the Arduino goes through one loop every 50ms while the python script would take around 100ms. Without going to far into detail, this would cause the plot to not be reacting to the ball’s movement in real time at all which was the entire goal.

This was fixed in two ways:

1. Clearing the serial buffer each loop in the python script which was throwing away data (not ideal)
2. Removing print functions which were outputting data to the python terminal. This was found accidentally when talking to another classmate about the problem. It turns out the python print method is very taxing with how it interrupts the loop. Removing this immediately made the plot smoother and more responsive

# Conclusion

In conclusion, there was a lot of challenges which came up to overcome. We had to manage our expectations and goals of the project to determine how we found solutions to the challenges weighing cost of supplies, complexity, and time management.

GitHub Link: <https://github.com/drizzt165/PID-Balance-Beam-Project>

# References