Nick Brannies, Cristian Garcia

PID COntrol visualization

ECE3301 - Final Project

Contents

[Abstract 2](#_Toc23155650)

[Introduction 2](#_Toc23155651)

[Experimental Methodology 2](#_Toc23155652)

[Experimental Results 2](#_Toc23155653)

[Challenges Faced 2](#_Toc23155654)

[Conclusion 3](#_Toc23155655)

[References 4](#_Toc23155656)

# 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:

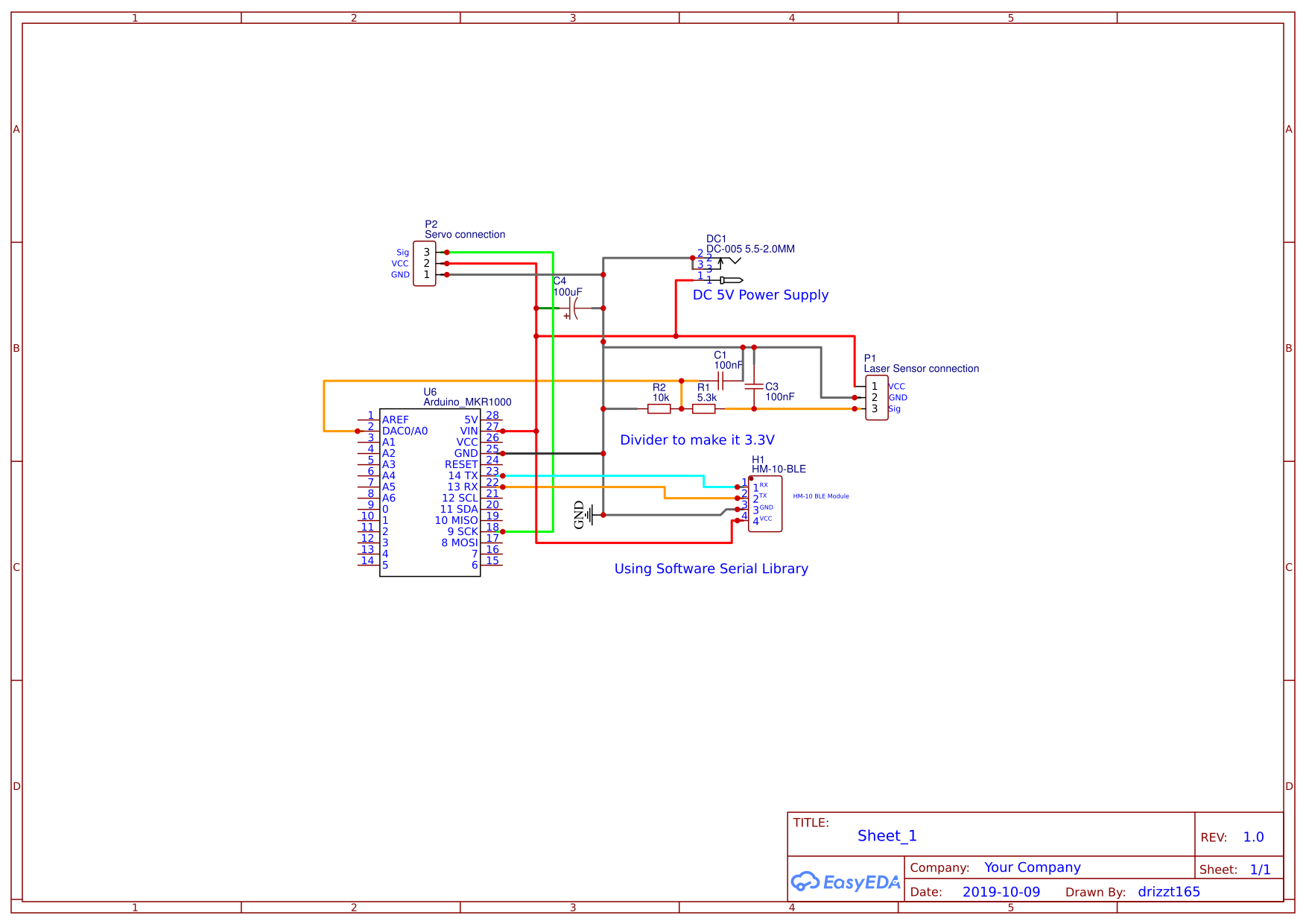
which represents separate discrete values.

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:

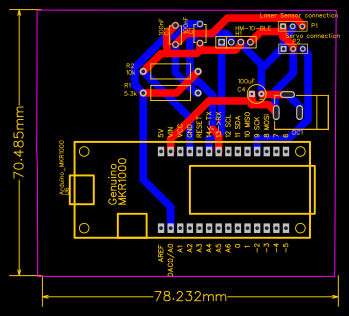
For our microcontroller, the Arduino MKR1000 which operates on 0 – 3.3V, we have a resolution of:

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 1 - Schematic of project as a whole



Circuit - PCB layout of designed circuit

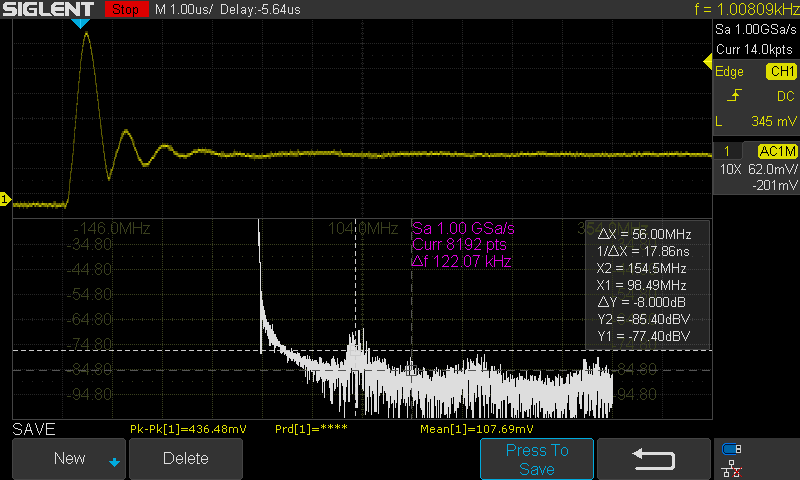
# Experimental Methodology

# Experimental Results

# 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.

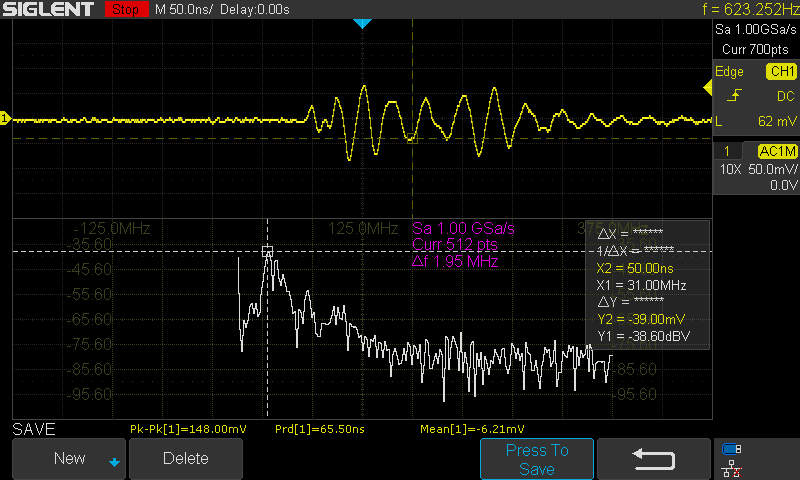
In the hardware side of things, the biggest challenge by far was implementing the infrared light sensor which outputs a voltage based on the distance an object is in front of it. The signal is supposed to be purely direct current (DC); however, we ran into an issue with it having a peak to peak voltage 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.



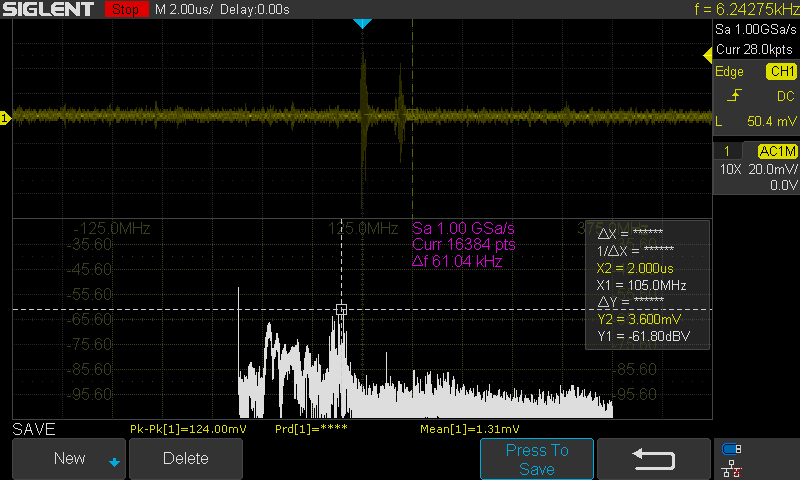
Scope Capture – Scope Capture showing a non-dc component of our signal before filtering

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.

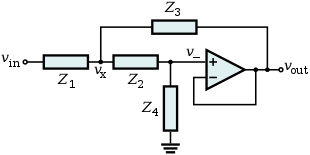


Scope Capture - Screen capture of signal after filtering



Scope Capture - 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 - Filter circuit schematic where V1 is the output signal of the sensor

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

# Code

## Main Arduino Sketch

## Python script for serial Plotting

# Conclusion

# References