Magnetic Levitator Arduino Project

Andrew Rippy

Department of Physics, Wabash College, Crawfordsville, IN 47933

May 2021

Abstract

The goal of this project was to create a magnetic levitator using a Arduino Microcontroller, by creating a circuit to control current through a solenoid, and creating a PID to accurately react given position data of the magnet to control the strength of the magnetic field.

1 Introduction

This project was an endeavor to explore what we had learned over the course of our semester in Electronics, by applying our knowledge in an exciting way. As the title of this report suggests, I created a magnetic levitator using the Arduino to control it. I utilized my knowledge of op-amps, DAC's, and transistors, in addition to my coding knowledge to create the levitator. I will go into the specifics of the circuit in my Model section, in addition to the idea behind the PID.

2 Model

The circuit consisted of a DAC connected directly to the Arduino, to control the current output. The DAC outputs 0-2.5 V to a op-amp amplifier with a gain of 2, to step the output up to 0-5 V. This amplification was required due to the subsequent follower op-amp requiring a minimum voltage of 1.7 volts to properly function. This follower op-amp had a much higher max current, which allowed greater flexibility in the current range. From this follower op-amp, a signal transistor stepped the current up to about 180 mA, and then a power transistor Darlington Pair further stepped the current up to the required range of 0-5 Amps. This current was then put through a solenoid coil with an iron core through its own separate power circuit, pulling directly from the power supply through the coil then to ground in a separate power circuit. A fairly problematic part of utilizing that much current is that things get **HOT**. To combat with this problem, a piece of aluminum metal was cut, and two holes drilled into it for each power transistor. The aluminum heat sink was fitted onto the transistors, and then placed into a beaker filled with cold water, in order to dissipate heat. A separate circuit for the HC-SR04 Ultrasonic Sensor connects directly

to the Arduino, that reads in the position of the magnet every millisecond. The magnet being levitated is a rod stuck into a piece of styrofoam for support, with a flat foam bottom so the ultrasonic sensor can accurately grab its position. The circuit diagram is included later in this report. Now with the hardware out of the way, the meat of the project comes in; the PID. PID stands for "Proportional-Integral-Derivative" controller, which as its name suggests, controls the circuit based on a proportional, integral, and derivative term.

$$u(t) = K_p e(t) + K_i \int e(t) + K_d \frac{de}{dt}$$

Where $e(t)$ is the position of the magnet at a given time, and $u(t)$ is how the controller is going to change the output current. Though not the most elegant solution, PID's essentially boil down to educated guessing and tweaking to get the terms K_p, K_i, K_d to a point the circuit can effectively control the magnet's position. Five position points were stored in a position buffer, where each point was a average of 5 points taken in succession. This was to cut down on the noise created by the ultrasonic sensor. Using this buffer and some numerical analysis, a four-point backward derivative stencil was applied of the form $[x_0, x_1, x_2, x_3]$, giving us the derivative at x_3. The stencil coefficients are as follows: $[-\frac{1}{3}, \frac{3}{2}, -3, \frac{11}{6}]$. In a similar way, a five-point Simpson's integral stencil was applied, of the form $[x_0, x_1, x_2, x_3, x_4]$, which gives the integral of the function from x_0 to x_4. The stencil coefficients are as follows: $\frac{1}{3}[1, 4, 2, 4, 1]$. The integral is with respect the optimal position. The proportional term scales from how far the current position is from the optimal position. What this essentially boils down to in layman's terms is this:

Proportional: How far am I from the optimal point?
Derivative: How fast am I moving?
Integral: How long have a been away from the optimal point?

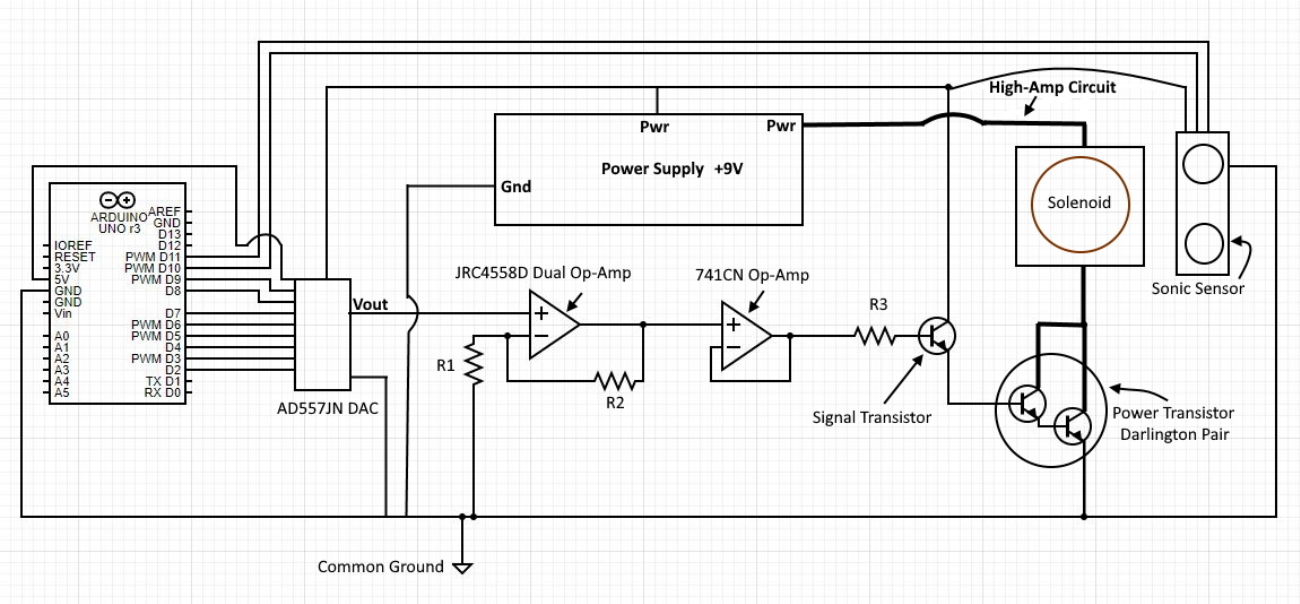

Figure 1: The Levitator Circuit

3 Experiment

3.1 Parts

The exact specifications of the hardware, with the circuit diagram for reference the following parts were used:

1. R_1, R_2, R_3 were standard $\frac{1}{8}$ watt resistors, with values 1kΩ, 2kΩ, and 100Ω respectively.
2. AD557JN DAC. This takes in a 5 volt power supply and outputs 0-2.5 volts based on the 8 inputs for each input. The exact specs are outlined **here.**

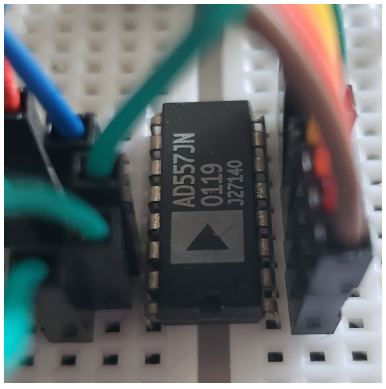

Figure 2: The DAC

3. JRC4558D Dual Op-Amp. This Op-Amp was used as an 2x voltage amplifier for the DAC output. The exact specs are outlined **here.**

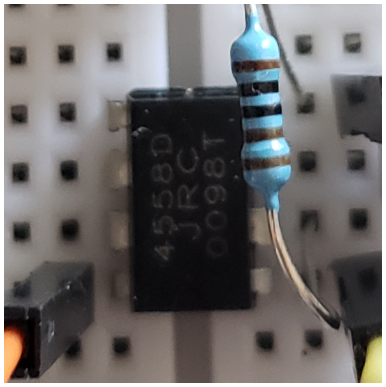

Figure 3: The JRC4558D Dual Op-Amp

4. 741CN Op-Amp. This Op-Amp was used as a follower to drive more current, with a max current of 40mA. The specs are outlined **here.**

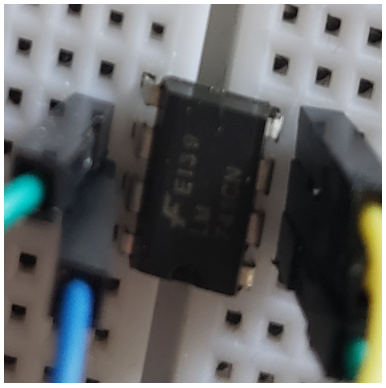

Figure 4: The 741CN Op-Amp

5. 2N3904 NPN Signal transistor. This was used to step the current up. The exact specs are included **here.**

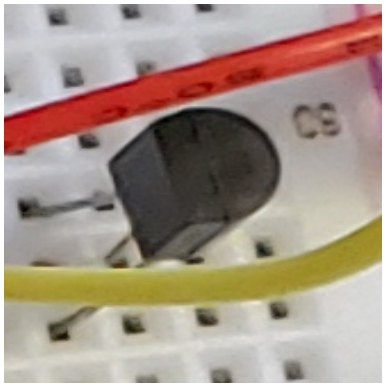

Figure 5: 2N3904 signal Transistor

6. TIP120 Power Transistors. These did the bulk of the current amplification, and drew at a maximum 5 amps of current. The exact specs for these transistors can be found **here.**

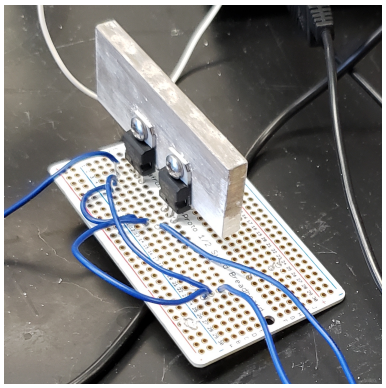

Figure 6: The TIP120 Power Transistor power circuit, soldered to its own separate board

7. HC-SR04 Ultrasonic Sensor. This was used to find the position of the magnet, with a resolution of 0.3 cm. The exact specs for this can be found **here.**

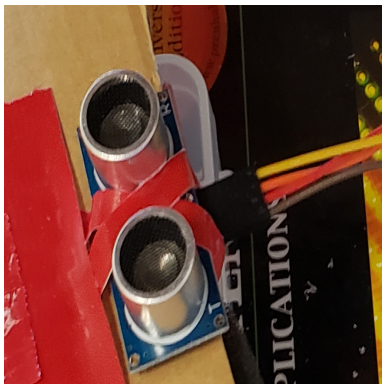

Figure 7: The HC-SR04 Ultrasonic Sensor

8. Leybold Solenoid with an iron core. The effective inductance of the coil with the iron core is approximately 4.4mH. The solenoid can be found **here.**

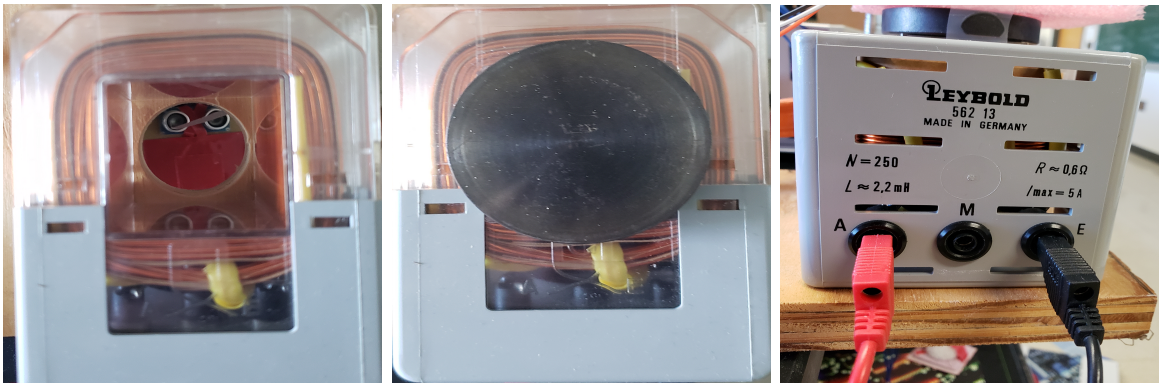

Figure 8: A view of the solenoid with no core, with the core, and a side view

9. PASCO Scientific Low Voltage AC/DC Power Supply. The exact specs can be found **here.**

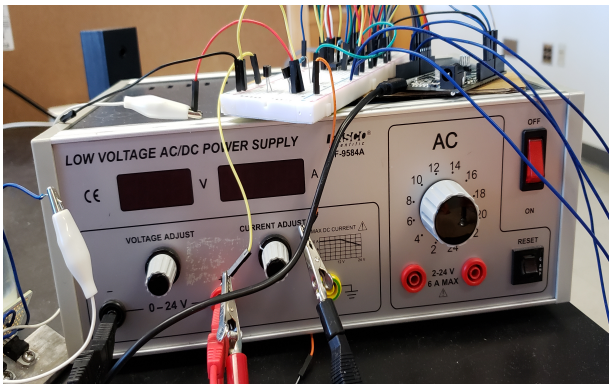

Figure 9: The PASCO Scientific Low Voltage AC/DC Power Supply

10. Adafruit METRO 328 board. The exact specs for this board can be found **here.**

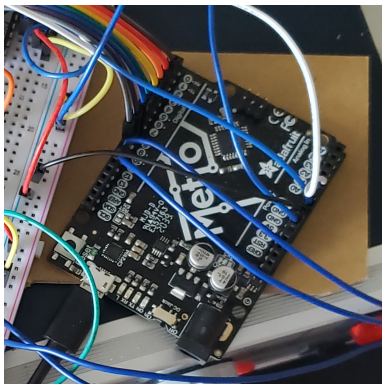

Figure 10: The Adafruit METRO 328 board

11. The magnet itself is the rod. It has a flat foam base to be accurately detected by the ultrasonic sensor, and a piece of styrofoam to stabilize the rod.

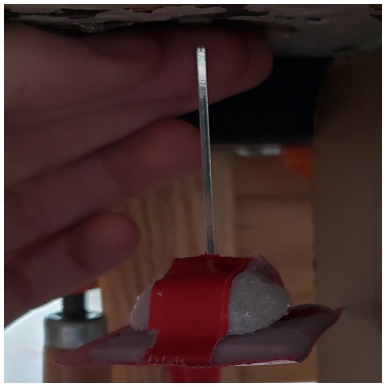

Figure 11: The magnet being levitated

3.2 Data

To confirm consistent control of the current using the Arduino, Current data for the outputs was collected. The current increased in a relatively linear manner with respect to the Arduino output, as shown below:

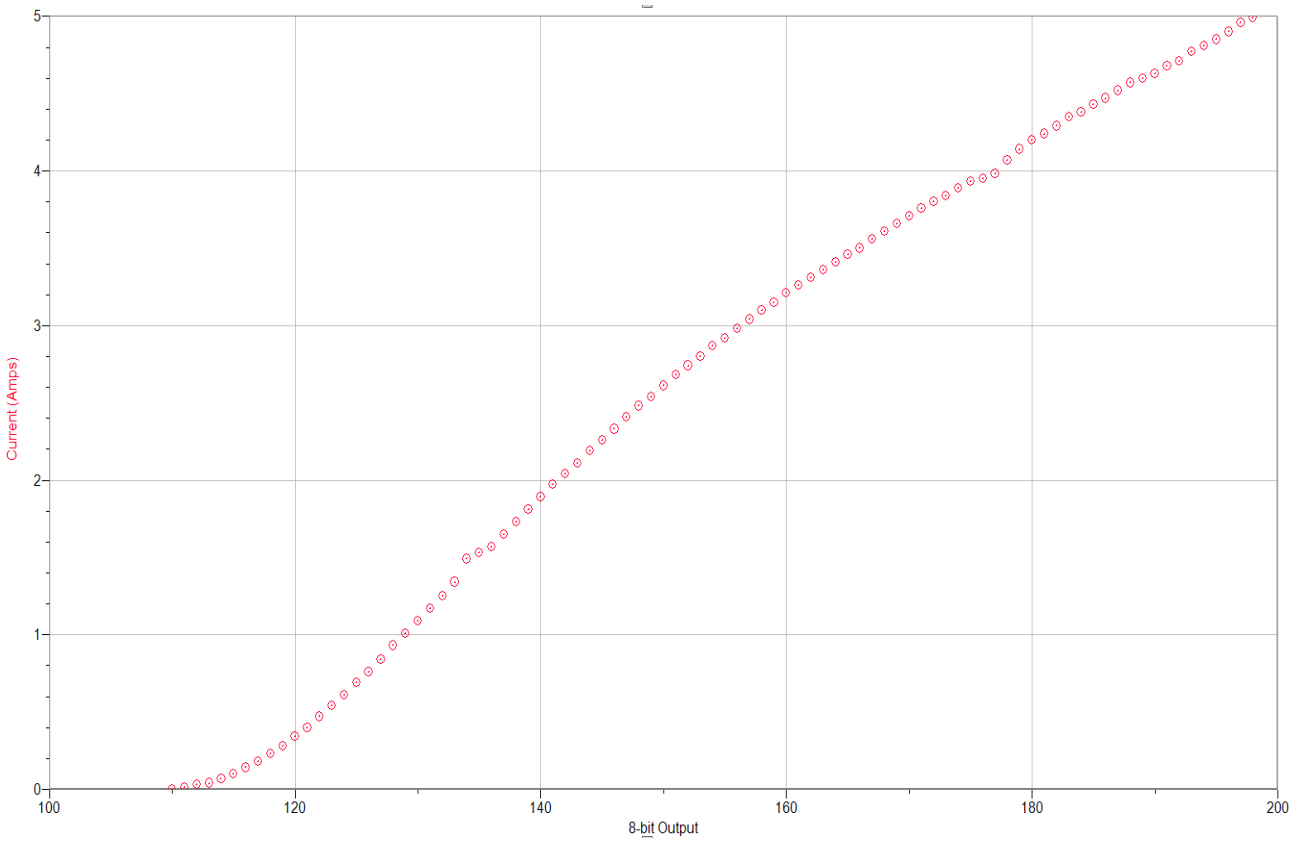

Figure 12: A graph of the Solenoid current vs. Arduino Bit output

The full area of movement for the magnet relative to the ultrasonic sensor was 0-14cm, (0cm being at the sensor, 14cm being stuck to the solenoid) and the noise of the position

measurement was found to be lowest around the range 11-14 cm. Using the current data in conjunction with this, a stationary point for the magnet was chosen to be at 3.5 amps, with a bit output of 170. This resulted in the stationary optimal position of the magnet to be approximately 12cm, within the "low noise" zone. This set-up is shown below:

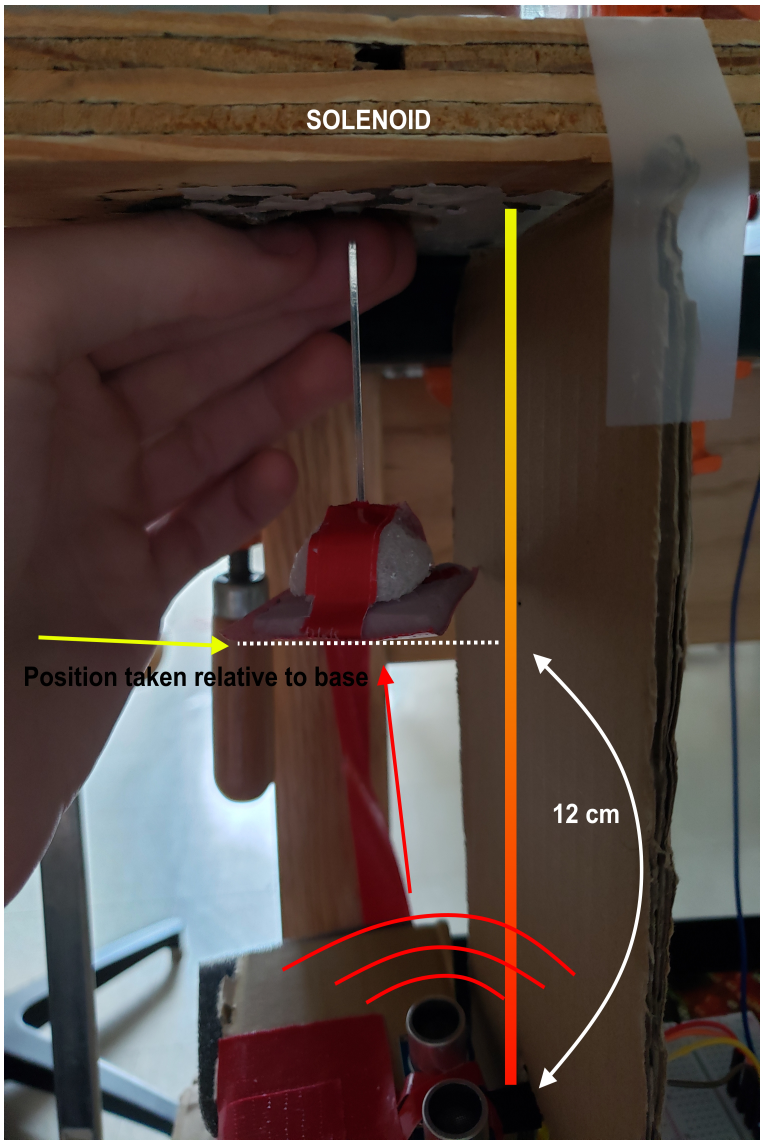

Figure 13: A picture of sonar-magnet setup

3.3 PID Arduino Code

```
#include "NewPing.h"
NewPing sonar(10,11,100);
int pins[10] = {2,3,4,5,6,7,8,9};
double buff[5] = {0,0,0,0,0};
```

```
void setup() {
  // put your setup code here, to run once:
  for(int i = 0; i<8;i++){
    pinMode(pins[i],OUTPUT);
    }
    Serial.begin(9600);
    pinMode(10,OUTPUT);
    pinMode(11,INPUT);
    digitalWrite(10,LOW);



    for(int i = 0; i<5; i++){
      updateBuffer(buff, samplePoints());

      }
}
int bits = 0;
int bits2 = 0;
int bits3 = 0;
int totalbits = 0;
double temp = 0;
double prevavg = 0;
double optimal = 12;

double total = 0;
void loop() {

  updateBuffer(buff, samplePoints());


  bits = (int) 2*(optimal-buff[4]);


  bits2 = (int) -derivative(buff);



  bits3 = (int) -integral(buff)*3;

    Serial.println(bits);
    Serial.println(bits2);
    Serial.println(bits3);
    Serial.println("");
  //Serial.println(buff[4]);
```

```
    totalbits = 170 + bits + bits2 + bits3;
    if(totalbits < 110){ totalbits = 110;}
    if(totalbits > 255){ totalbits = 255;}
    if(buff[4]>14){totalbits = 100;}
   setBits(totalbits);


       //3.5 amps = 12 cm
}
```

```
int offset;
int tmp;
void setBits(int num){
  offset = 1;
  //Serial.println(num);
  for(int i =0;i<8;i++){

    digitalWrite(pins[i], num&1);

      num = num >> 1;

     }
     //Serial.println("");

  return;
  }

double* updateBuffer(double* buff, double newval){

  buff[0] = buff[1];
  buff[1] = buff[2];
  buff[2] = buff[3];
  buff[3] = buff[4];
  buff[4] = newval;
```

```
  //Serial.println(newval);
  return buff;
  }

double avg = 0;
double samplePoints(){
  for(int i =0;i<5;i++){
      avg += (sonar.ping()*0.0343/2);
      }
      avg /= 5;

  return avg;
  }




double integral(double* buff){
  double sum = 0;
  double h = 1.0;
  sum += buff[0]-optimal;
  sum += 4.0*(buff[1]-optimal);
  sum += 2.0*(buff[2]-optimal);
  sum += 4.0*(buff[3]-optimal);
  sum += buff[4]-optimal;

  sum = sum*(h/3.0);

  return sum;
  }

double derivative(double* buff){
  double sum = 0;
  sum+= (11.0/6.0)*buff[4];
  sum+= -3.0*buff[3];
  sum+= 3.0/2.0*buff[2];
  sum+= -(1.0/3.0)*buff[1];
  return sum;

  }
```

Explanation of the code: The top of the code begins by including NewPing.h, which is a helper file to make using the ultrasonic sensor easier. I then define a new sonar, with output ping pin 10, and input ping pin 11. Next, I define a pins array, which stores the 8 pins which

will be the DAC inputs. The buff array contains the 5 most recent position points taken by the position sensor. The setup() function initializes the digital pins to be in the correct setting, and sets the serial port to 9600 for debugging purposes. It also takes the first 5 position points to populate the buff array. The actual loop itself is quite small, since most of the code is contained in their own respective methods. These methods are:

1. samplePoints(), which grabs 5 raw position points from the sonar and averages them to reduce noise, and then outputs the average.

2. updateBuffer(), as its name suggests, updates the buffer. It throws out the oldest point, shifts everything over to the beginning, then inserts the newest point at the end of the array.

3. derivative(), as its name suggests, takes the derivative at the current point given the position buffer. It uses the stencil outlined in the model section

4. integral(), as its name suggests, takes the 5-point integral using the stencil outlined in the model section.

5. setBits() takes in a number 0-255, and converts this to its 8 bit binary, then sets the respective output pins for the DAC to reflect the input value.

The various print statements that exist in the code were used for debugging purposes. The loop itself, as outlined in the PID section of the model, updates the position buffer with the current point, calculates and weights the proportional, derivative, and integral terms different to achieve the desired outcome. Once each of the 3 parts had been computed, they were added as an offset to the base term of 170 bits. There are lower and upper limits to prevent the output from breaking.

4 Analysis

Now with the hardware and software out of the way, we can look at and discuss the results of this project. The project was success, with the magnet levitating with the above code. You can see it levitate in this video **here!** By using the above weights; 2 for proportional, 1 for derivative, and 3 for the integral, and an optimal point of 12cm, we were able to get the magnet to levitate. Averaging the position smoothed out the noise further, and still allowed the Arduino to take position fast enough to react. In the video, I am not supporting the magnet at all, but rather preventing it from spinning, which I found was a problem, as it quickly caused it to fall out of place due to spinning messing with the position readings. Another problem was the heat of the power transistors. As explained in the data section, the current stayed consistent, *provided* that the power circuit remained cool. The longer the circuit was on for, the more likely heat would play a factor in the solenoid coil, but the transistor pair as well. The water would quickly heat up after a few minutes, and would have to be moved to cool it down once more. As the circuit began to heat up, the current began to skew from the expected values, rendering the PID inaccurate.

5 Closing Remarks

Overall, I am quite proud of this project. I feel it displayed my knowledge of what I had learned in electronics with the creation and implementation of the circuit. Despite the noise from the ultrasonic sensor initially giving me issues, I was able to work around this and get the magnet to levitate, even if only for a few seconds. This project could have been further improved on with a more accurate position sensor, however I was unable to find any better Arduino compatible ultrasonic sensors. This circuit could also be improved by implementing a better cooling system for the transistors, to control the heat output by the circuit. Likewise, the PID coefficients could have been further tweaked to provide longer more consistent results, and this would just take further testing and time to implement.

Acknowledgements

I acknowledge the support of the Wabash College Physics Department. Thanks for a great year despite COVID. See you next year!