Artifact ID:	Artifact Title:	
ASID	Automation Subsystem Circuit Design	μsphere
Revision:	Revision Date:	
005	7 APRIL 2021	
Prepared by:		Checked by:
Seth White		Dylan Howlett
Purpose:		
Design, Testing, and Implementation of the Automation Subsystem		

Revision	Revised by	Checked by	Date	Description
001	S. White	T. Christensen	18 FEB 2021	Initial Release
002	S. White	R. Anderson	22 FEB 2021	Added Purpose, Specs, BOM, Eagle Schematic
003	S. White	D. Howlett	25 FEB 2021	Added Summary, Made Text More Specific
004	S. White	T. Greene	8 MAR 2021	Added Button Kill Switch Implementation and Pull-Up Resistor
005	S. White	D. Howlett	7 APRIL 2021	Added PCB and Adapter sections

Purpose

To automate the system, we need an actuated DC solenoid valve. To do this we need to control the DC solenoid valve with a microcontroller connected to a driver circuit. This will allow the system to be cycled at precise and accurate timing.

Summary

The automation subsystem consists of an Arduino Uno, printed circuit board (PCB) driver circuit as an Arduino Shield, solenoid valve, custom 3-D printed Arduino to Solenoid Adapter, and 13V DC power supply. The whole subsystem has a mass of 100 g and a volume of 0.1 ft3, far below our desired values shown in the Automation Subsystem Requirements Matrix artifact. Current draw from the Arduino peaks at 9mA, and current draw of the whole Automation Subsystem is near 1A, allowing the system to work for a minimum of 1000 uses.

Specifications

- < 10mA from Arduino
- 12V +- 10% across load
- 1 A +- 10% current draw from power supply

Initial Design

I chose to use a 12V Electric Solenoid Valve because we want to control the system with an Arduino. An Arduino is convenient because can be powered by a 12V DC source and use that as a power source for the driver circuit and solenoid. Because we are using the Arduino only one power source is needed: a wall plug in 120V AC – 13V DC adaptor. I chose a 13 V DC adaptor so the solenoid wouldn't need to pull as much current to function properly and is still within specifications. I made a driver to control the solenoid with the Arduino so we don't burn out the microcontroller by drawing too much current. The level shifter provides a higher voltage which is necessary to switch the power MOSFET. The power MOSFET is necessary because the solenoid requires greater than 1 amp. In Figure 1 you can see the schematic for the circuit. There is also a switch implemented with the Arduino that has the capability to turn off the system at any point (see S1 in Figure 1.)
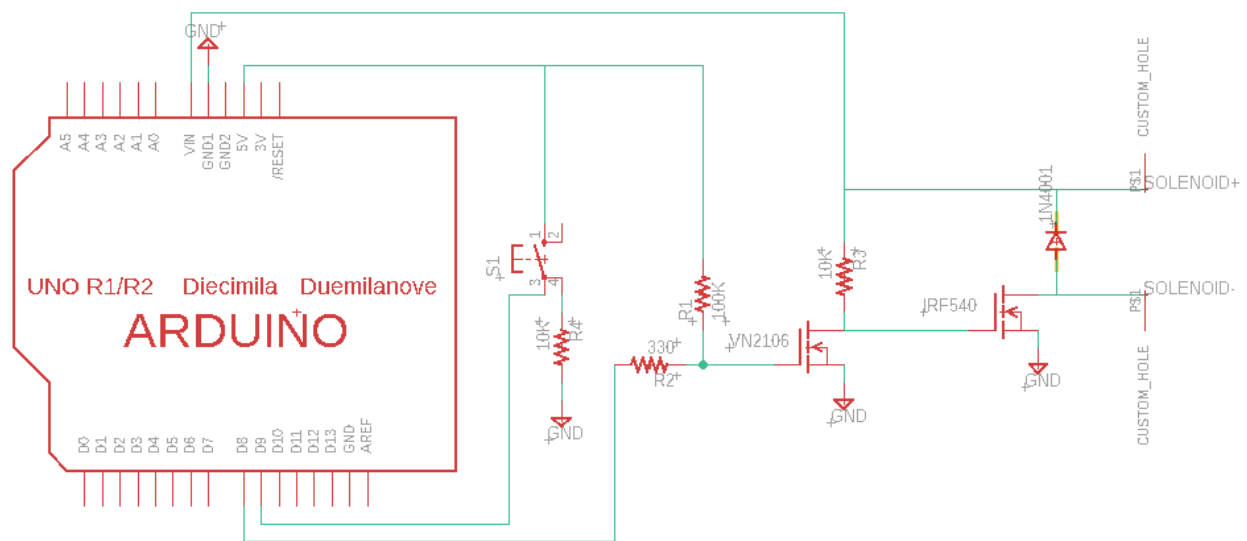

Figure 1: Solenoid Driver Eagle Schematic.

From the schematic I generated a bill of materials as seen in Table 1. The total cost of the driver circuit using the ECE shop is $6.21. The cost of the whole automation subsystem is $34.71.

Table 1: Bill of materials for the driver circuit.

Quantity	Value	Device	Package	Part	Description
1	N/A	10-XX	B3F-10XX	S1	OMRON SWITCH
1	100KΩ	R-US_0207/12	0207/12	R1	Resistor
1	10KΩ	R-US_0207/12	0207/12	R3	Resistor
1	10KΩ	R-US_0207/12	0207/12	R4	Resistor
1	1N4004	1N4004	DO41-10	1N4004	Diode
1	330Ω	R-US_0207/12	0207/12	R2	Resistor
1	IRF540	IRF540	TO220	Q1	nMOS 100V; 28A; 0,077Ohm
1	VN2106	VN2106	TO237	2N2106	nMOS 60V; 0,5A; 7,5Ohm
1	N/A	PCB	N/A	2021-SD3	Solenoid Driver Board

I simulated the design in LTspice (Figures 2-5) to confirm the circuit would act as predicted.
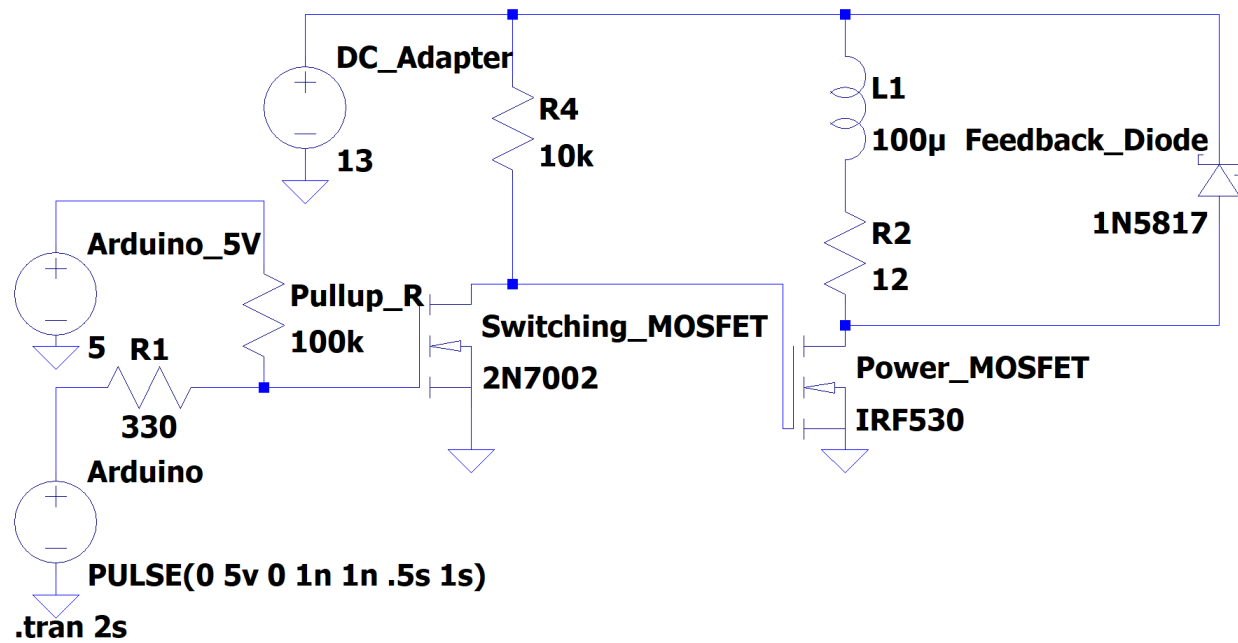
Figure 2: Solenoid Driver Spice Schematic used in the Spice simulation. Components L1 and R2 represents the Solenoid valve within the circuit.

Design Results

I ran the tests by simulating the Arduino with a pulse 5V in, switching every half second. The current draw from the Arduino is <10mA.
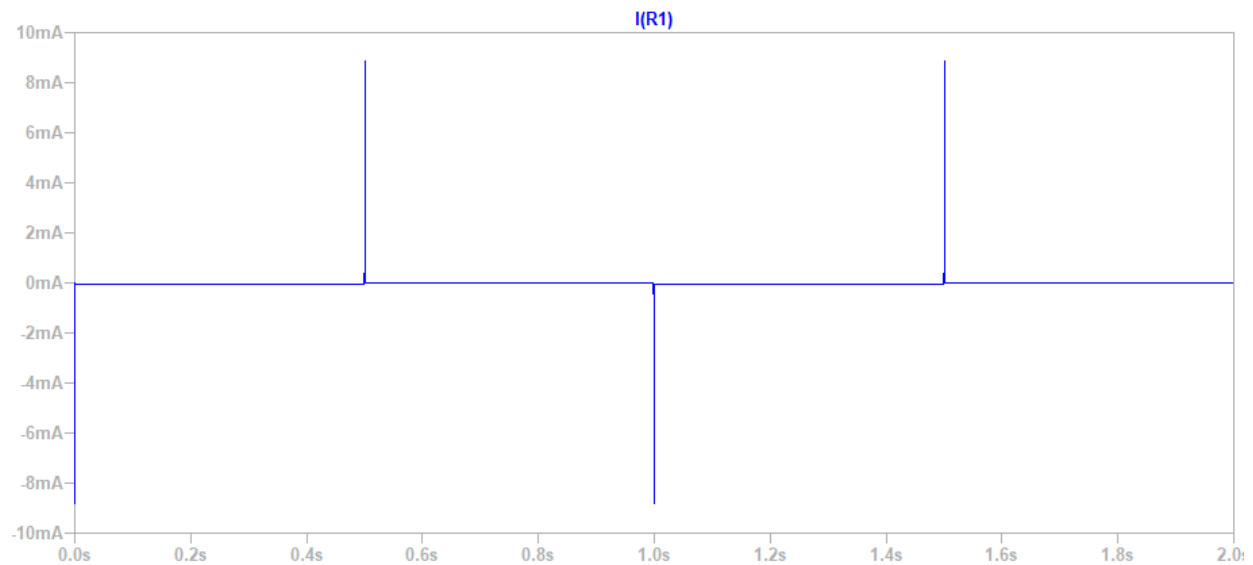

Figure 3: Driver Input Current Analysis. Current drawn from the Arduino by the driver circuit over time.

The total current draw of the system as simulated is 1.07A. This allows the 13V 1A DC power supply to power the whole automation subsystem.
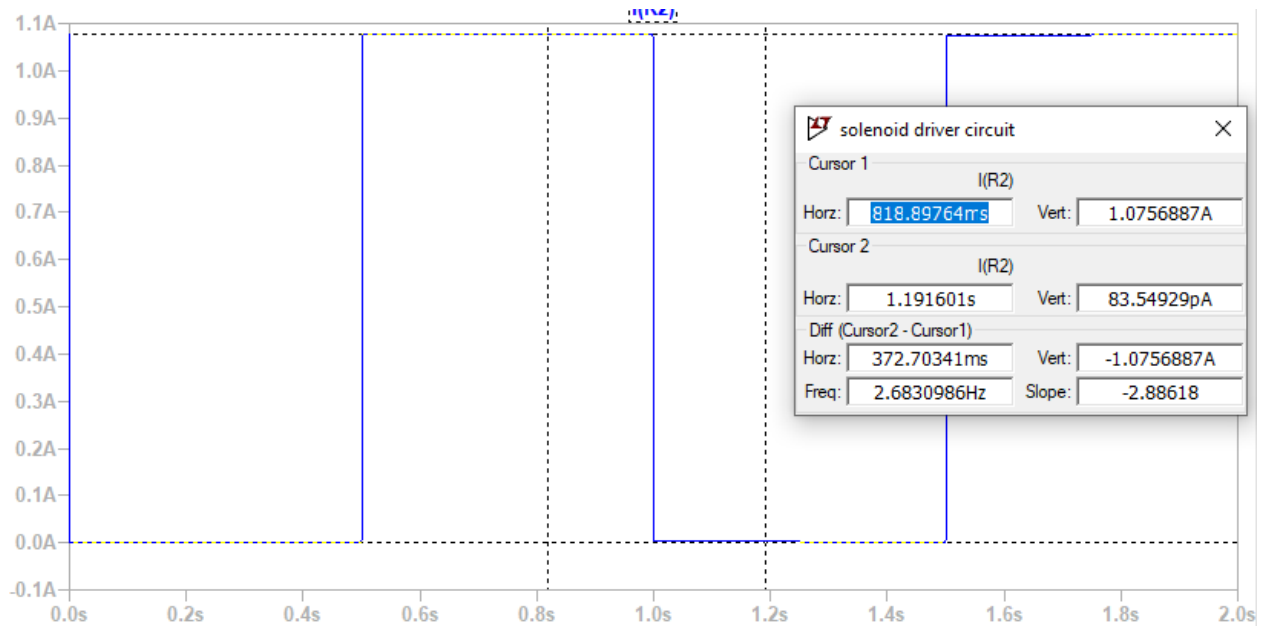
Figure 4: Current draw from the 13V 1A power supply over time. Maximum current draw is 1.07A.

As seen in Figure 5, the total voltage drop across the Solenoid valve is 12.91V. This value is below the rated 13V of the power supply.
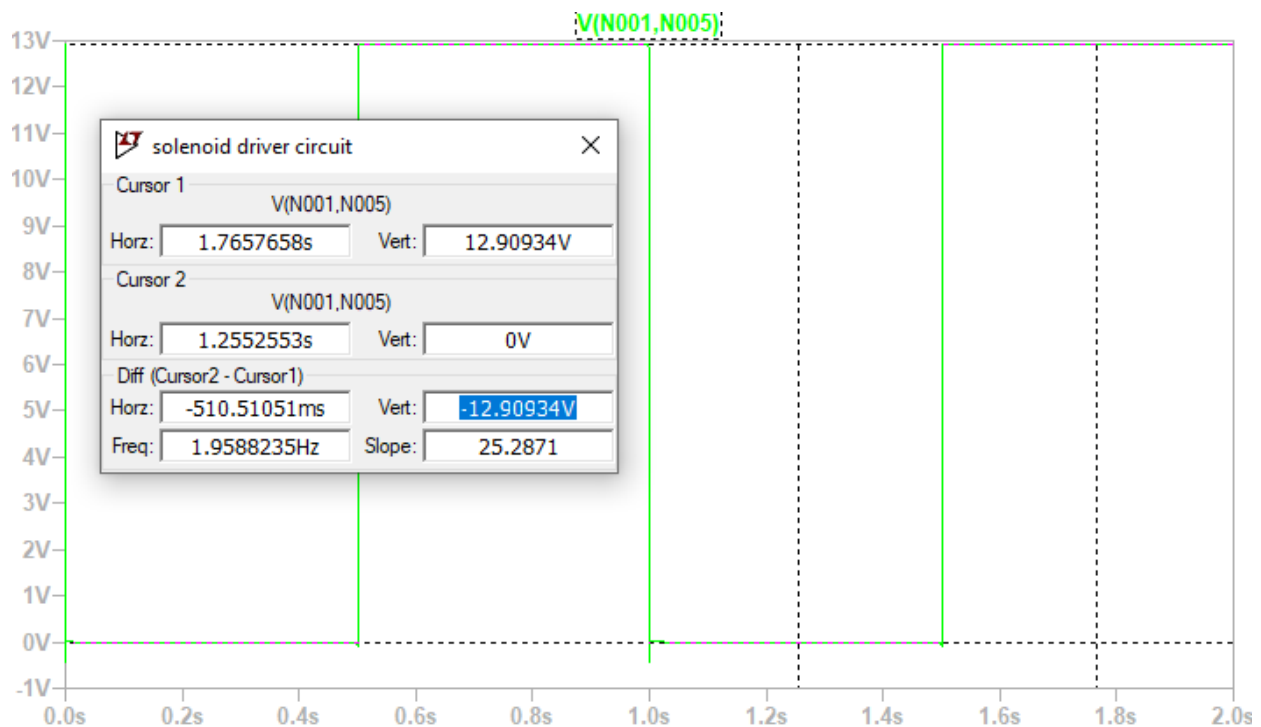

Figure 5: Solenoid Voltage drop of 0V-12.92V over time.

Prototyping

Parts Used:

- Switching MOSFET is VN2106
- Power MOSFET is IRF540N
- Tactile Button
- Solenoid valve is U.S. Solid Electric Solenoid Valve- 1/4" 12V DC Solenoid Valve Plastic (Nylon) Body Normally Closed, NBR SEAL
- Proto Board
- Arduino Uno
- 120 AC to 13V 1A DC power supply

The circuit was soldered onto a protoboard and was made into a face shield for the Arduino. The circuit works as expected. It is an inverting circuit so when the Arduino outputs high the valve will close and when the Arduino outputs low it will open.
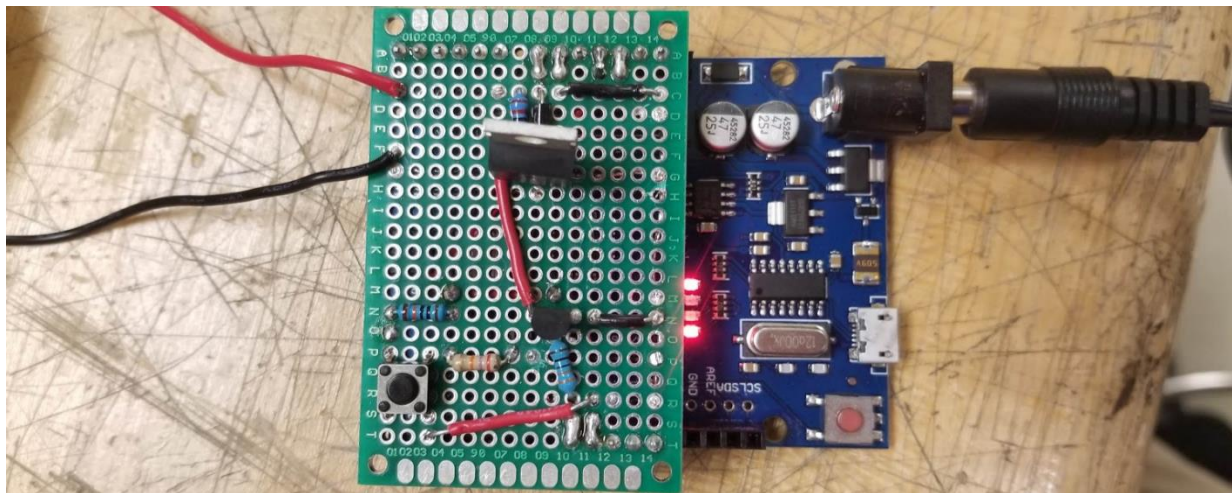

Figure 6: Board Layout with the soldered protoboard and Arduino.

Automation Control

I coded the Arduino so that when you press the button on the protoboard it will open the solenoid air valve for 10 minutes. The button will act as a kill switch for the valve if it is pressed at any time during the 10 minutes and will reset the timer. If you press it again it will open the valve for 10 minutes. Refer to Appendix I for the code implementation.

PCB Design

I decided to fabricate the board on a PCB to make the shield easier to replicate and make shield look better. I used the data in Figure 1 to fabricate a board layout as seen in Figure 7.
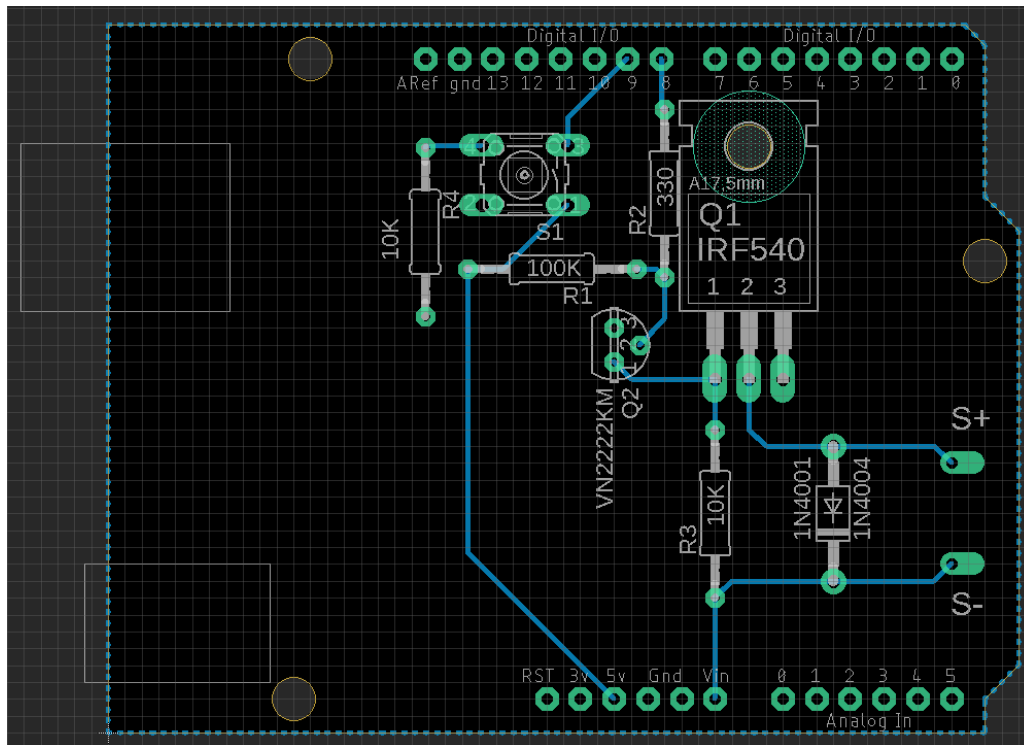
Figure 7: PCB layout.

With this layout, I can fabricate the shield easily. Figure 8 shows the ground layer of the circuit.
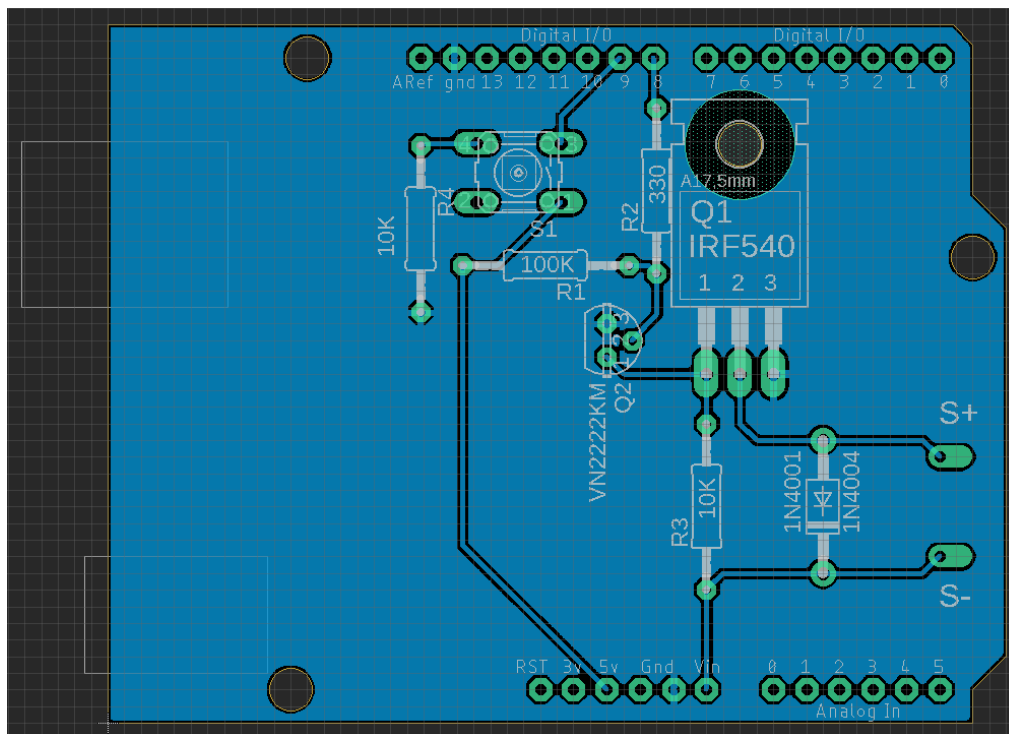

Figure 8: PCB ground layer.

I then modelled the board in CAD to have an idea of what it would look like.
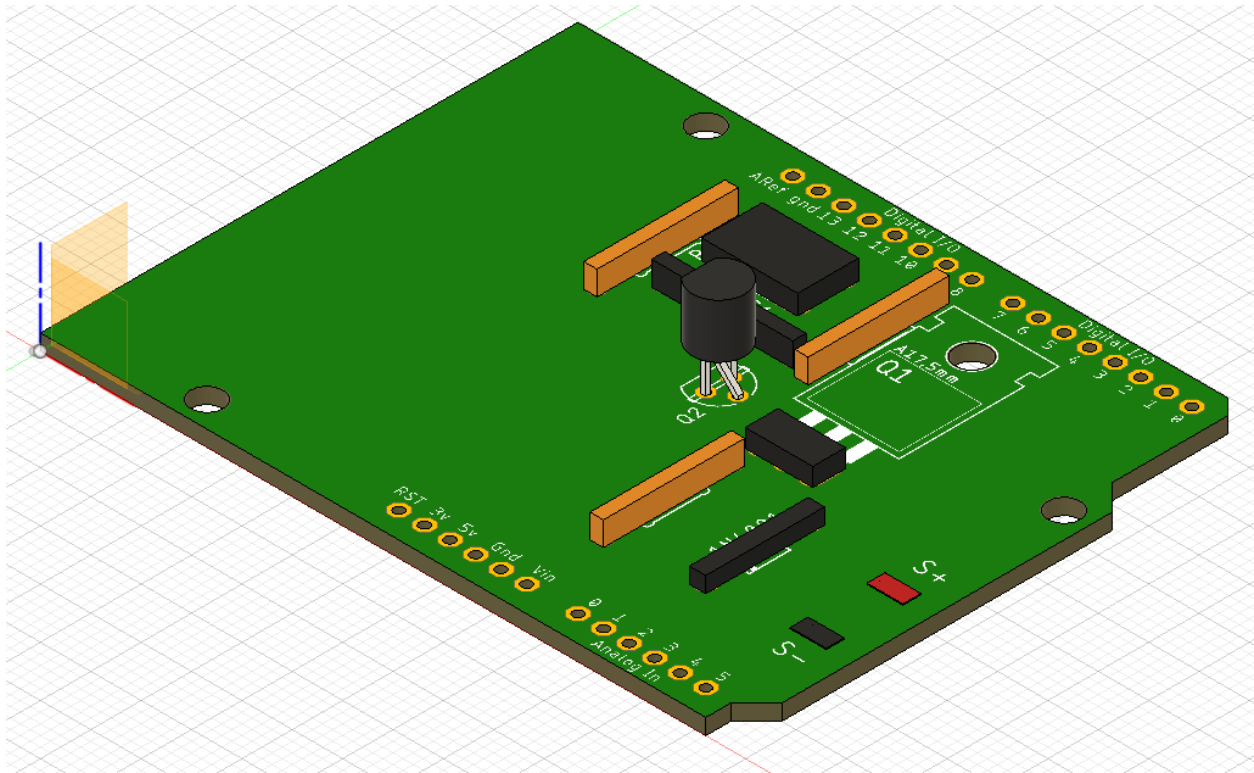

Figure 9: 3-D model of PCB.

The PCB was printed with the ground layer on top because the PCB printer available only prints one side and I needed to solder header pins onto to board for it to interface with the Arduino Uno. This made it function properly. The circuit elements were all soldered onto the bottom with the button on top for easy access. I also soldered on pins that allow the PCB to be attached to the Arduino as a shield. To increase safety and decrease probability of shorting the circuit I put a clear coat finish on the top (ground plane.) I plugged PCB shield into the Arduino and ran it with the code in Appendix I and it ran properly. The fabricated PCB can be viewed in Figures 10-11.
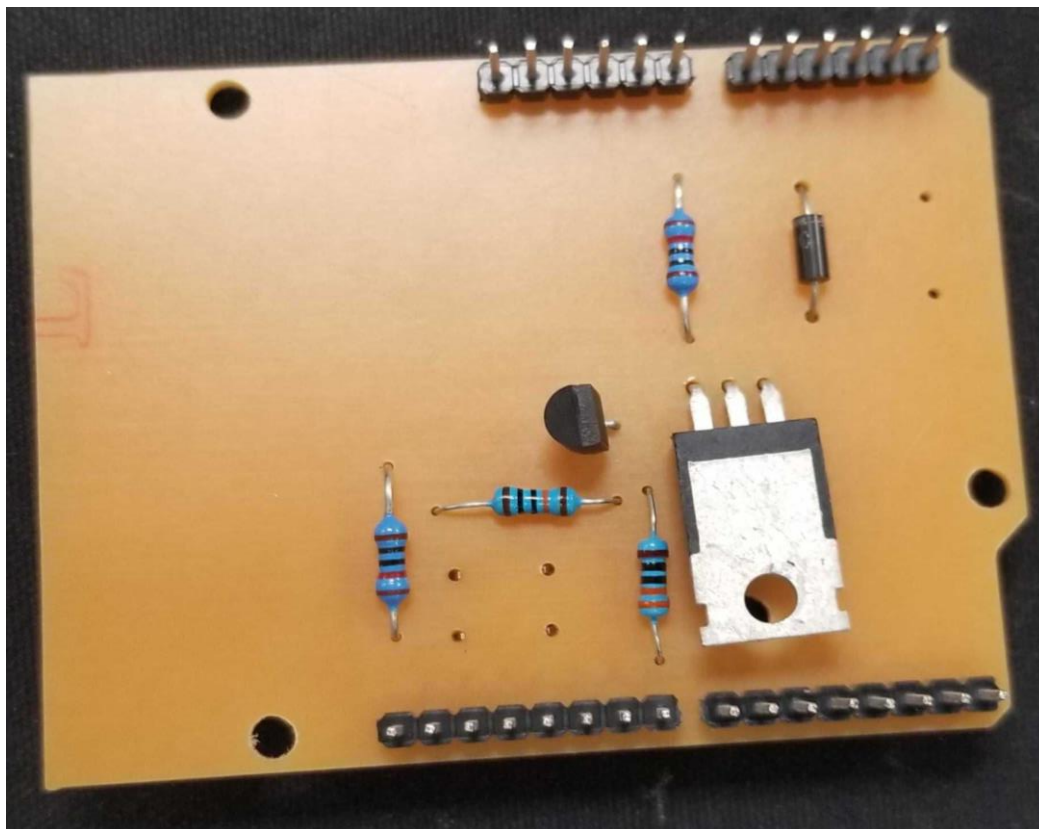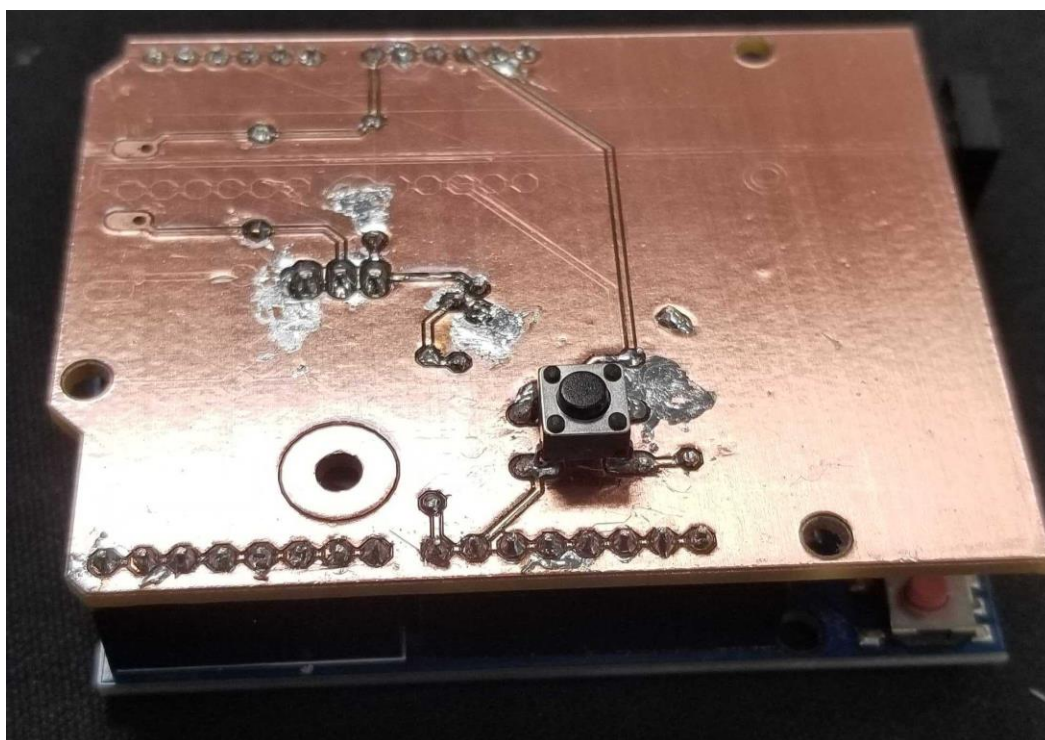
Figure 10: Bottom-side view of printed PCB.

Figure 11: Top-side view of printed PCB mounted onto the Arduino.

Arduino to Solenoid Adapter

We designed a custom 3-D printed part to attach the Arduino to the DC valve. We got measurements provided by the manufacturers of the DC valve and Arduino Uno to acquire the data needed such as dimensions and screw holes. The part designed can be seen in Figure 11. For more information on the drawings, refer to: Electronics_Mount - Sheet1.
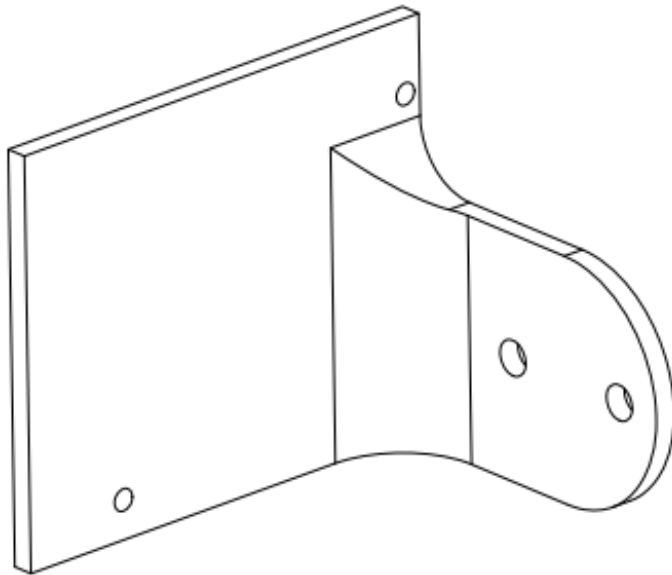
VIEW A
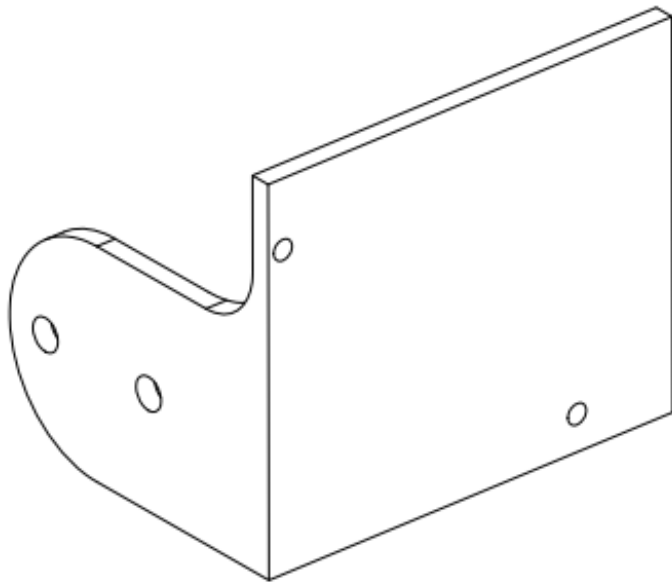
VIEW B

Figure 11: Arduino to Solenoid Adapter

We printed the adapter in ABS so that the heat from the valve will never become an issue. We then screwed the parts together and it functioned as designed. Make sure to print the part so that the strands run through the elbow, not aligned with it. The first adapter made broke because of that. Figure 12 shows the final product. For more information on assembly, refer to: Automation Subsystem – Sheet 2.
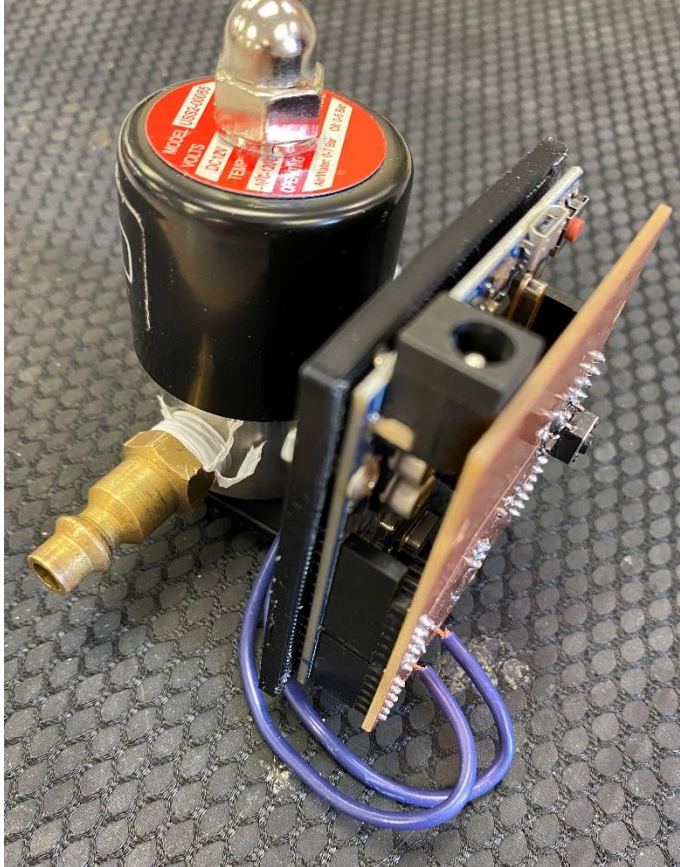

Figure 12: DC Valve with Arduino and Shield Attached with custom 3-D printed part.

Conclusion

The board works as designed for driving the solenoid valve. Never program the board while it is plugged into the AC-DC adapter. It will burn out the voltage regulator on the board if done so. Do not leave the valve open for more than 8 hours, you could damage the coils. It is an inverting driver so at low the solenoid will open and at high it is closed. There is a start/stop button implemented that increases safety and control. Print the PCB with the ground layer on top if your PCB printer cannot do through-hole fabrication. Make sure to print the part so that the strands run through the elbow, not aligned with it. This system will make testing and controlling the valve easy.

Appendix I

Arduino Program code:

```
const int outputPin = 8;
const int buttonPin = 9;
const long int t_final = 5000; // 5 minutes, 500 is 30 seconds, 1000 is 1
minute


bool run_driver = 0;
int buttonState = 0;
long int timer = 0;

void setup() {
  pinMode(outputPin, OUTPUT); // sets the digital pin 8 as output,
  // default is HIGH b/c of a pull-up resistor
  pinMode(buttonPin, INPUT);  // sets the digital pin 7 as an input,
  // default is LOW b/c of a pull-down resistor
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);
  // check if the pushbutton is pressed. If it is, invert the value for
// run_driver. (if it's on, turn it off and if it's off, turn it on)

  if (buttonState == HIGH) {
    if(run_driver == 1) {
      timer = t_final;
      run_driver = 0;
    }
    else {
      timer = 0;
      run_driver = 1;
    }
    while (digitalRead(buttonPin) == HIGH){
      delay(50);
    }
  }
  // If run_driver flag is raised keep the system on. Otherwise keep it off.
  if (run_driver == 1 && timer < t_final) {
    digitalWrite(outputPin,LOW);
    timer++;
  }
  else {
    digitalWrite(outputPin,HIGH);
    run_driver = 0;
  }
  delay(50);
}
```