# Automated Pet Care System

ECE 4220

Final Project

Kyle O'Day

(kfoy82)

May 15, 2015

**Abstract:**

For my project I designed and developed a system that provides the basic capability to feed and water household pets. There are many households in the United States that keep cats and dogs as companion animals. While people greatly enjoy the company of their pets, busy lifestyles can make it difficult to provide consistent food/water schedules. A system that can aid the owner in automating the sustenance needs of their companions works to the benefit of both the owner and the animal. It was the goal of my project to show that such a system could be fairly easily designed and implemented using microcontrollers.

**Introduction:**

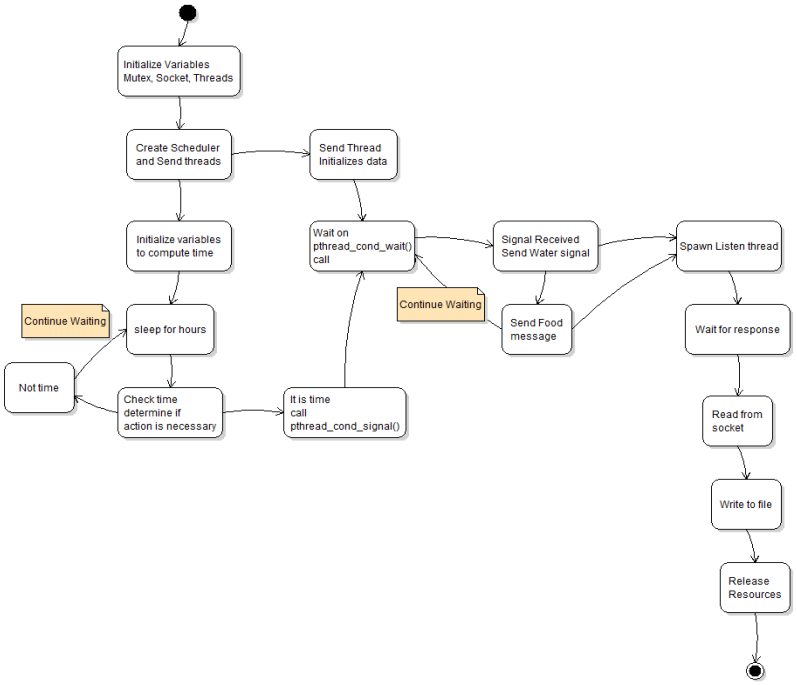
Caring for one's animal friends can become a chore over time. Many pet owners will readily admit that they do not dedicate enough time to caring for their pets. I myself am one of these pet owners. With a full load of classes and a part time job it is difficult for me to provide food to my pets on a consistent schedule. Another problem I have encountered is the obligation to constantly refill their water dispensers so that they are sustained and happy pets. These problems were the motivation for my project. I designed and built a rough prototype of a system that could refill a water reservoir and dispense food for pets. The system is comprised of a Raspberry Pi 2, Arduino Uno, Adafruit Arduino WiFi shield, solenoid valve, DC motor, Sainsmart water sensor, two breadboards, and various transistors, diodes, and resistors. The idea behind the project is that an Arduino can schedule pet care tasks and transmit instructions to an Arduino microcontroller over a wireless network to act on the instructions. The goals of my project were to communicate wirelessly between an Arduino and Raspberry Pi, control multiple components including a sensor on the Arduino, and to schedule times of the day for tasks to occur on the Pi.

**Background:**

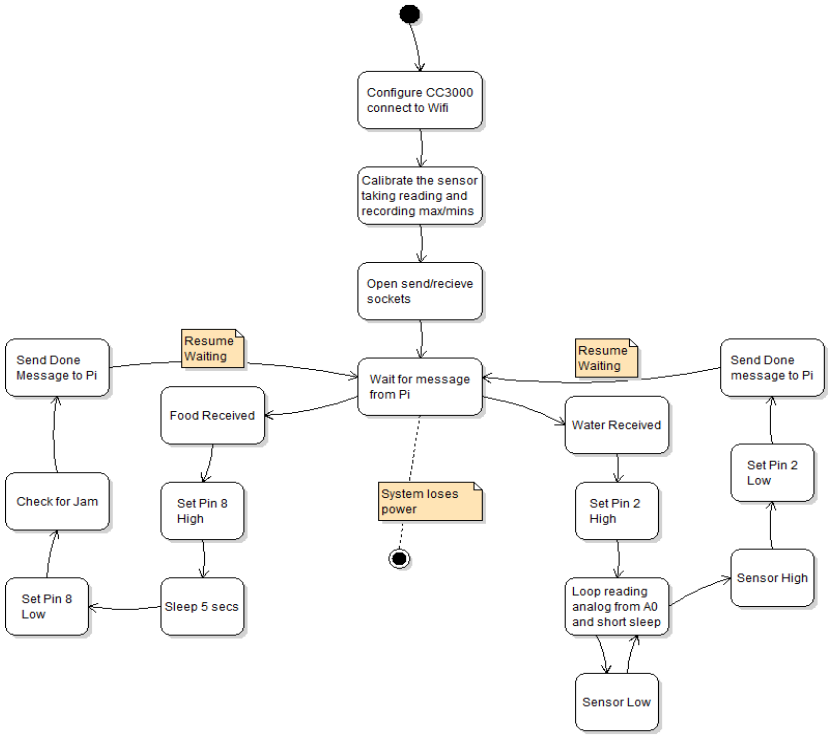
This project is inspired by systems that already exist on the market today. Gravity powered water dispensers for animals that consist of an upturned reservoir over a water dish are very popular among pet owners. I was unaware of pet food dispensers until I began my research on this project, but I was certain that they most exist. Food systems available on the market are fairly expensive and many reviews for them comment on their unpredictable reliability. Most of these work with a rotating exterior shell that covers pre measured amounts of food. The cover rotates at certain times of the day determined by the owner. My system could be used to replace both of these common tools and provide better results. Instead of having to refill the water reservoir the solenoid valve can do it automatically and the sensor will prevent overflows. The food system design reliability would be dependent on the mechanical system designed around it. While this design was not an objective of my project I do not think it is unreasonable to think it could be as reliable as dispensers currently available. This system could be used in any household that keeps companion animals to aid both the owner in saving time, and peace of mind knowing their pets are cared for. It would also benefit the animal because they would be able to maintain a steady feeding schedule and never run out of water to drink.

**System Description/Implementation:**

The system I designed is very simple. A Raspberry Pi acts as a client and an Arduino Uno as a server. The Arduino is equipped with a Wifi shield to allow the two controllers to communicate over a wireless network. The Pi program is a simple client that opens a UDP socket on the same port as the Arduino and then spawns three threads. The first is a scheduler that determines when a command should be sent to the client. The second controls outgoing messages through the socket, which command will be sent on each call. This thread waits on a pthread_cond_t variable for a signal from the scheduling thread to begin. After sending the message this thread spawns a child thread to listen for the response from the Arduino and log the times and any error to a text file. The Send thread then returns to waiting on its condition for more instructions from the scheduler. The Arduino sets ups its sockets for listening and sending to the Pi. It calibrates the water sensor which is connected to analog pin A0 and then enters a loop waiting for commands from the server. When a message is received conditional statements determine which action has been requested and then reacts accordingly. The Solenoid valve is powered by a 12V 1A source and is connected to the drain of a common source, power MOSFET transistor. The gate of this transistor is connected to digital pin 2. When water is requested pin 2 is set high which allows current to flow through the MOSFET and the valve to open. A loop checks the water sensor using an analog read on pin A0, when this value reads higher than the calibrated max, when then sensor contacts water, the loop breaks and the pin 2 signal goes low, closing the valve. A diode across the terminals of the valve prevents the inductive load from sending back current into the Arduino. The motor is controlled in a similar way but is controlled through pin 8 sending a high voltage to the base of a BJT. A timer allows the motor to run for 5 seconds, and a randomly generated number simulates the event of a jam. After action is complete a message is sent to the client to inform it of completion and a possible jam. The pi then records the time and message into a text file. The server then awaits new messages.




Raspberry Pi State Diagram




Arduino State Diagram

**Experiments and Results:**

I performed testing of my system in stages. First I assembled the circuits for the motor, sensor, and valve. I then tested controlling these with the pins of the Arduino. I then attached the wifi shield and tested communication between the Pi and Arduino. I started with very simple client/server programs. This is the part that caused me the most trouble. At first I could send messages from the Pi to the Arduino but I could not get messages sent back to the Pi. To test that the messages were received by the Arduino I used the serial monitor to print and observe them. Once communication was finished I then attached my circuits to the Arduino and modified my code for both components to send/interpret specific messages. The next chore was setting up thread synchronization and scheduling on the client. I spent a lot of time trying different methods to control the client. I already knew that the Pi did not have real time capability but I did not realize that there would be no easy ways to implement scheduling. In the end I was unable to achieve a satisfactory schedule and used sleeps to wait until it was time to send a command. This was the only failed objective of my project. I was able to successfully implement three threads and use mutexes and condition variables to synchronize their tasks. Testing this part was very slow and required starting both systems and observing the toggling of the valve/motor and testing that the sensor was registering water and closing the valve. I had to test endlessly to ensure the tasks were synchronized and did not fall into a state of deadlock. In the end I was satisfied that I had adequately tested and achieved two of my three objectives.

**Discussion and Conclusions:**

I learned a lot while attempting to create this project. I was able to send commands from my Raspberry Pi and interpret them on the Arduino which would then activate the necessary components. I observed the toggling of my motor/valve and that my sensor was correctly detecting water. The biggest problem I encountered was in network communication. The wifi shield I purchased used a TI CC3000 chip for its wifi communication. Due to this it came with a specially modified version of socket.h and other necessary libraries. These did not come with documentation so I was left to sift through forums and code to learn how to create my sockets and send/receive from them. I eventually gave up on UDP send from the CC3000 and tried TCP. With this I ran into the opposite problem, I could send to the Pi but could not get a complete message to the Arduino. All it would receive was partial messages. I tried for hours to troubleshoot this but eventually found a forum post that provided information on UDP send. Using this information I went back to UDP for communication. I was then able to finally achieve two way communication between the two controllers. This entire process took many hours and several days of work. Another major problem I encountered was with setting up and calibrating my sensor. The company that made it had no documentation on its use and I found limited resources online. At first I was always receiving a high signal from it. After experimentation I determined that I needed a pulldown resistor from the output signal to ground to ensure the signal was low when it was supposed to be. I used a 10 kOhm resistor for this. Examples of calibration were readily available online and that part was relatively easy. The biggest thing I learned was the limitations of scheduling when a controller is not real time capable. I was able to get the time of day from the network but the infrastructure needed to create and schedule real time tasks was nonexistent. After a lot of research I sadly admitted defeat and resorted to sleeps and querying the time to decide when a command should be sent. This implementation is far from satisfactory and is a major limitation of the system. If I could re do this I would either use a real time controller or attach a real time clock to the Pi and install a real time capable kernel on it. I was surprised at how easy controlling pins was on the Arduino. Functions are already built to handle this and no special methods such as bit masking were required. I was also surprised at how setting up sockets on the Pi was so similar to the methods used in lab assignments. This makes sense since both run on linux but I expected more difficulty in programming that part of my system. I believe that while my project was not a complete success I learned a lot about system design and limitations of these embedded systems.

**Appendices:**

**Code –** See Files provided with report.

**Wifi Shield Info**

Vendor Storefront: http://www.adafruit.com/product/1491

Tutorials: https://learn.adafruit.com/adafruit-cc3000-wifi/cc3000-library-software

Library Downloads: https://github.com/adafruit/Adafruit_CC3000_Library/archive/master.zip

**Circuit Components:**

BC547 NPN Transistor 1x

IRLB8721 N-Channel Power Mosfet 1x

2 kΩ Resistor 1x

220 Ω Resistor 2x

10 kΩ Resistor 2x

1 kΩ Resistor 1x

N-Type Diode 2x

LED 1x

12V 1/2’’ nominal plastic solenoid valve

5V DC motor

Water Sensor - http://www.sainsmart.com/sainsmart-water-sensor-free-cables-arduino-compatile.html

**Diagram:**

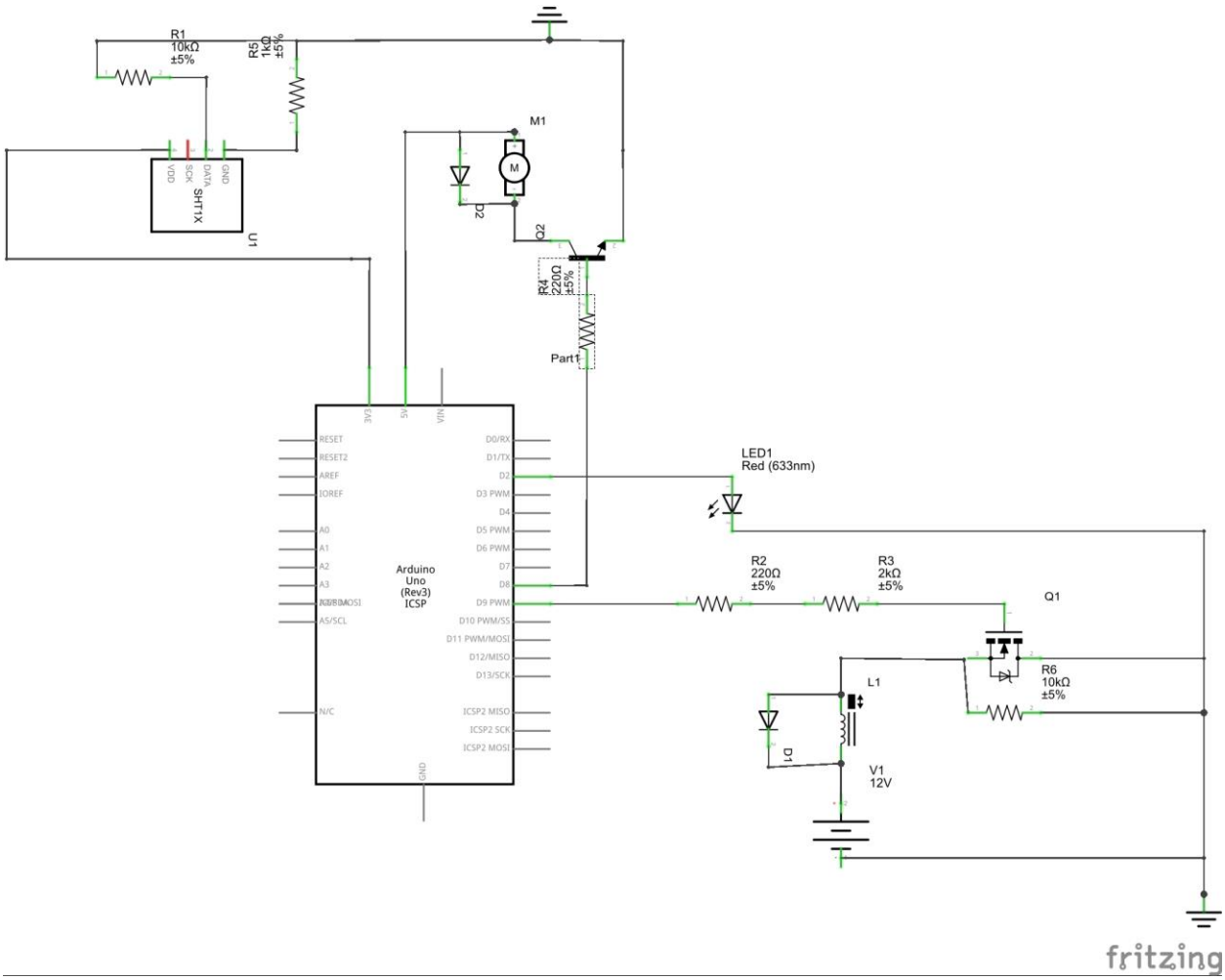
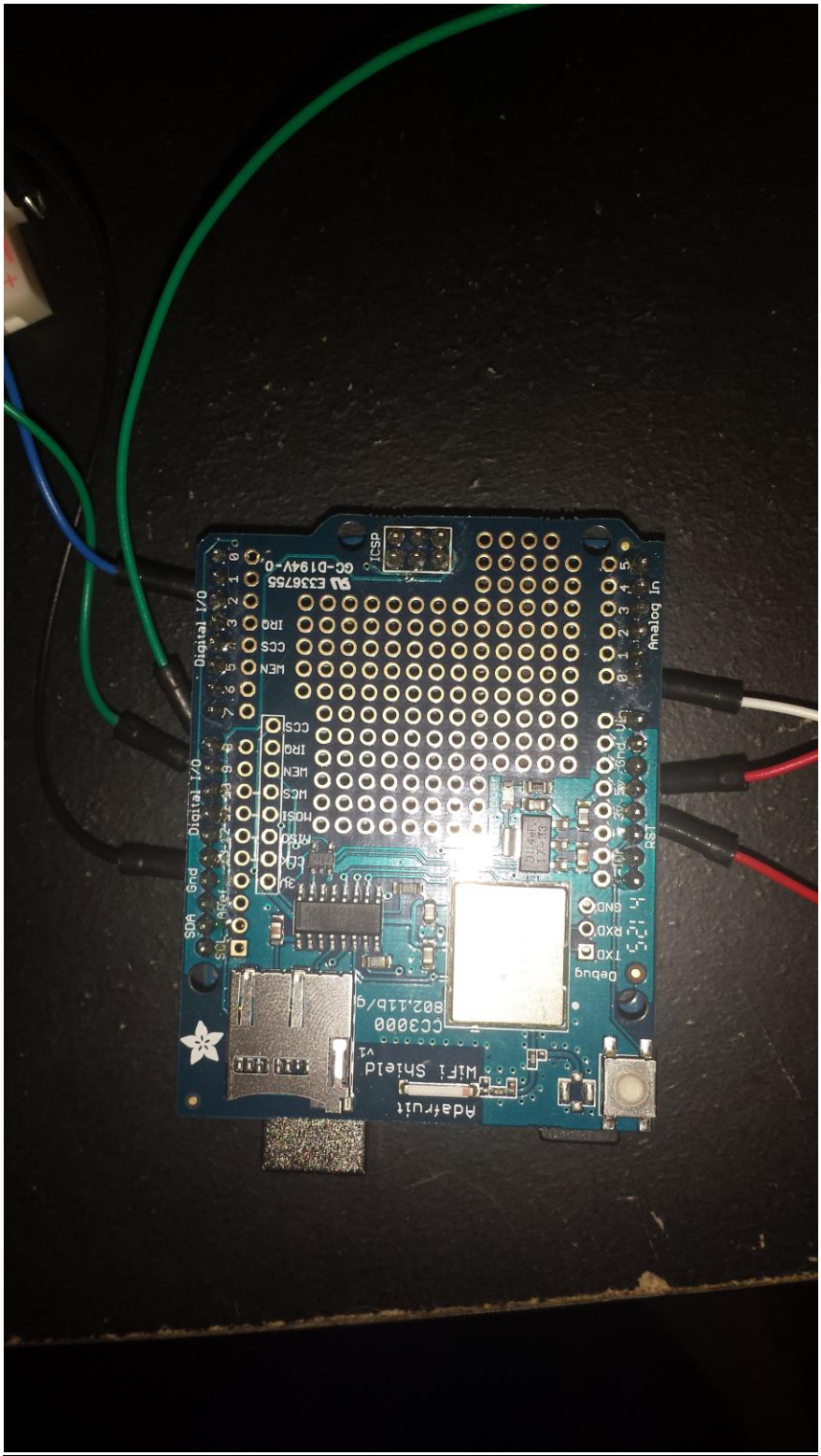
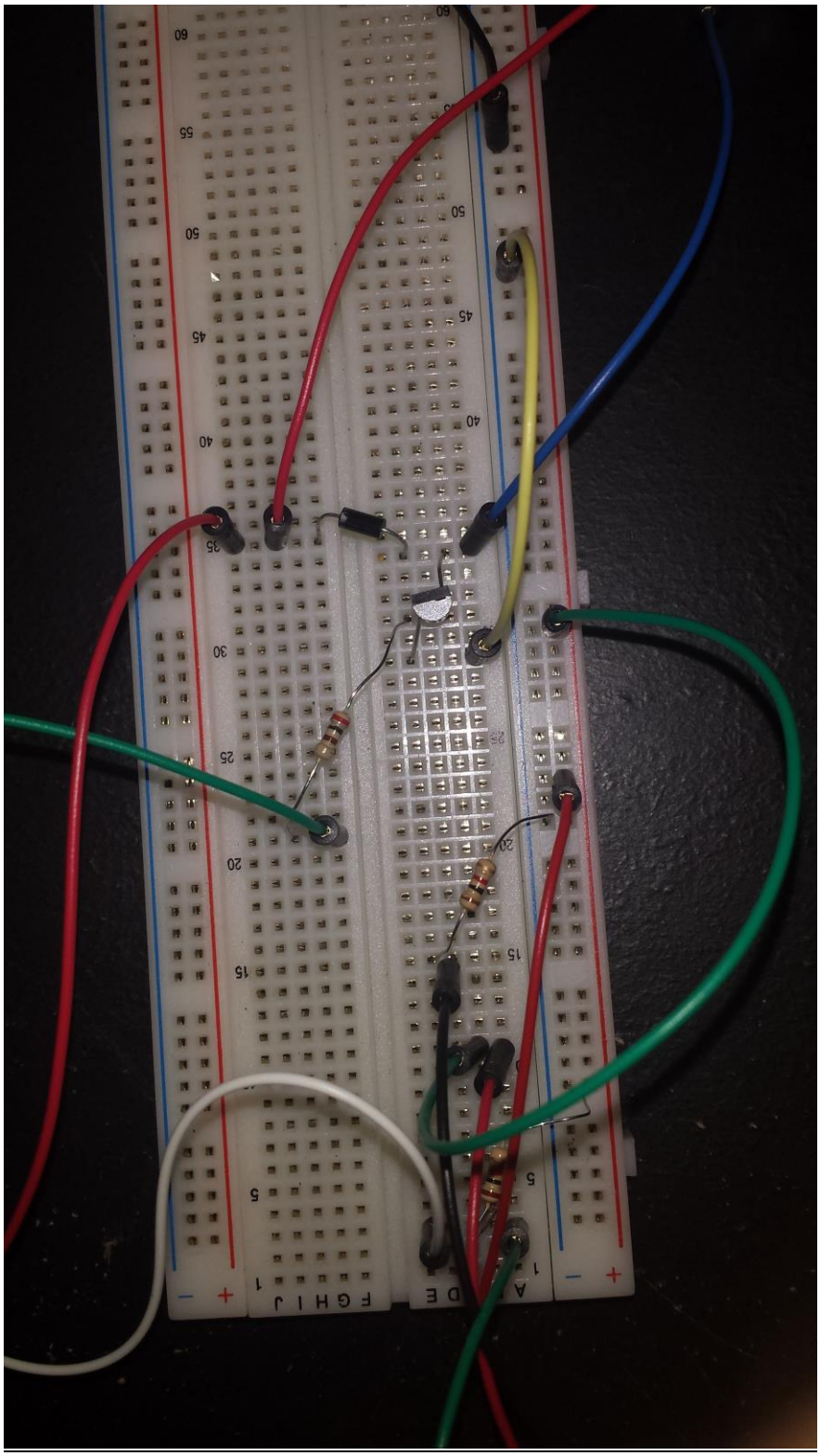



Diagram of Circuits

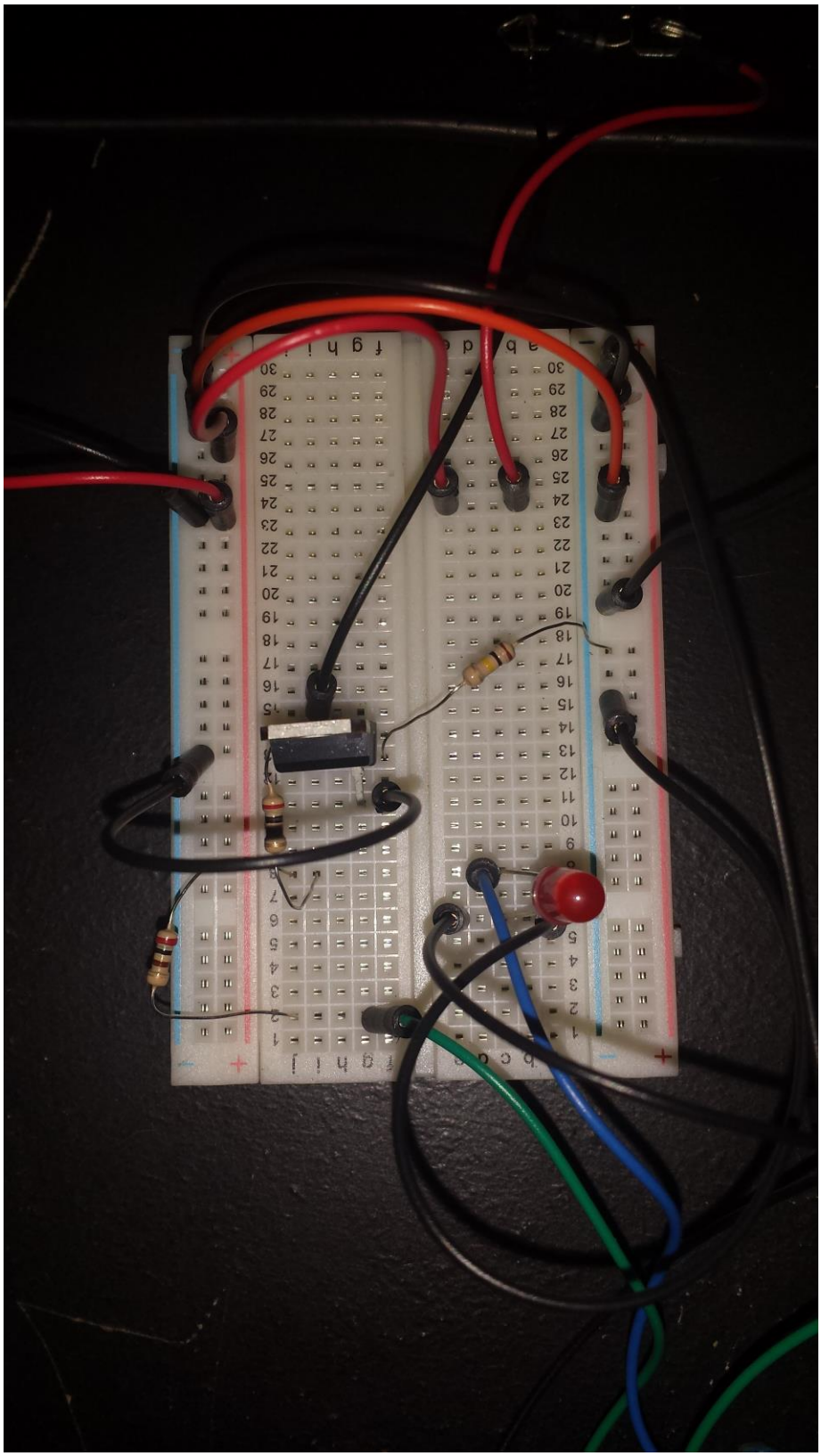
**Photos:**




**Arduino**



**Sensor/Motor Circuit**



**Solenoid Valve Circuit**