EE-4183/EE-4283 Senior Design Capstone Project
Final Report

Team JADE

By: John Lee, Anfernee Zhao, Damian Dziedzic

Date Due: 5/12/2018

Lean Startup Investigation

During the fall semester we decided to explore the idea of a smart drill. The concept was to have a simple attachment to a drill that could determine drill depth and notify you when you have changed mediums. Ideally, this device could also tell you the specific type of material that you're drilling through with a frequency profile of different materials.

Value Proposition:
"We present a device that guides people when drilling by detecting changes in mediums and predicting hidden materials, with a simple add-on feature to any drill."

Our target customers were large scale industrial drillers, amatuer DIY home improvement, and general construction companies. The key features for this device are attachability, simple to use, and have a wireless link. Our initial customer discovery, shown in figure 1, revealed that many people would like our device for these reasons:

- "Saves me time and money"
- "Less errors would be made"
- "Can improve safety and minimize damage"

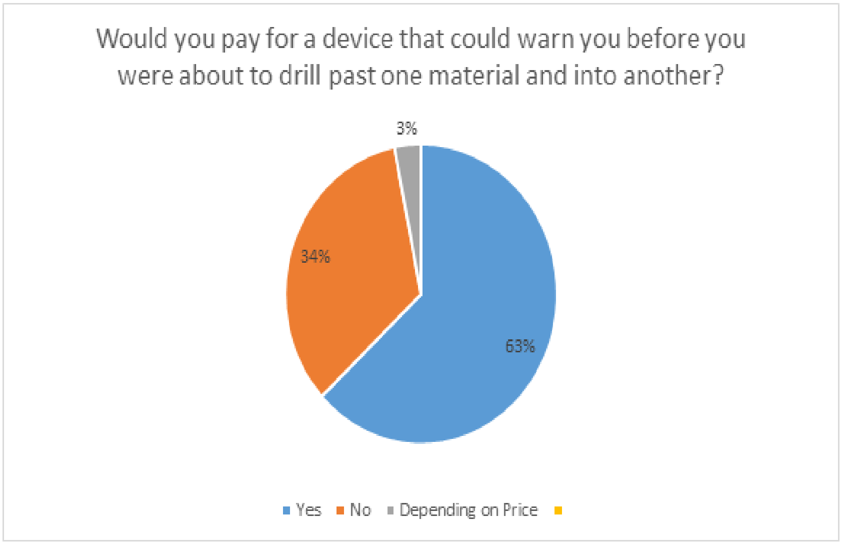


Figure 1

The more experienced craftsmen that we asked did not think much of our product because it would be of no use to them due to their experience. However, those that had little experience tend to think our product is a good idea. The most common complaints about power drills is shown in figure 2 below. Noise was the number one complaint that was brought up, but this is irrelevant to our product because it does not address this.

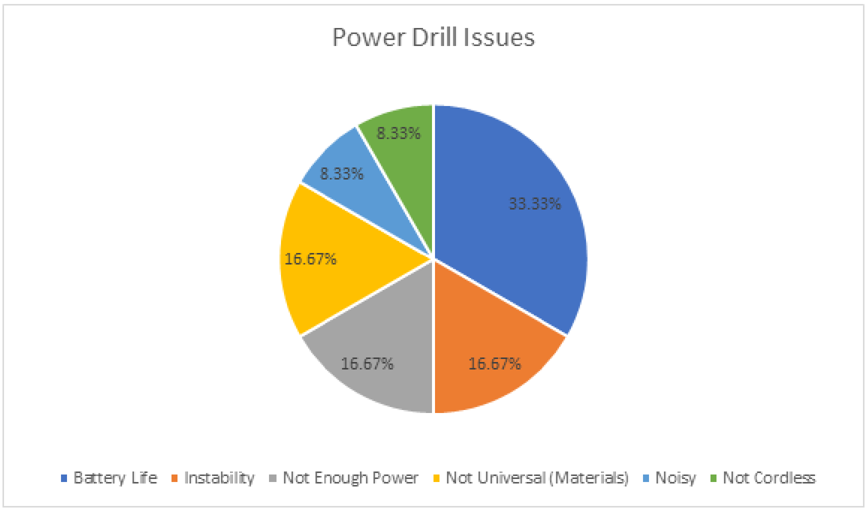


Figure 2

Our MVP design was an O-ring attachment to the chuck of the drill that had an LED indicator that would tell you whether or not you drilled through a medium. Figures 3 and 4 show the original concept of the design and figure 5 shows the O-ring that we 3-d printed with the LED attached.

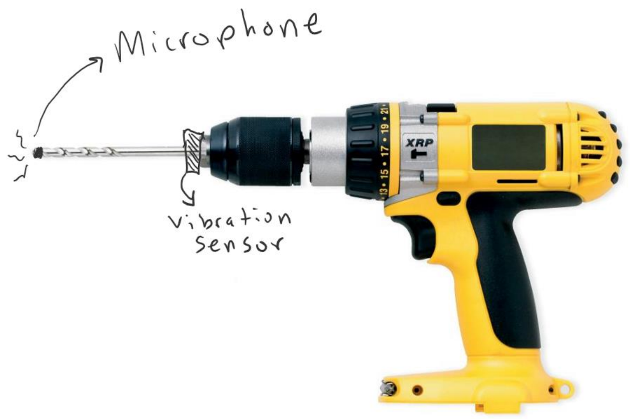


Figure 3

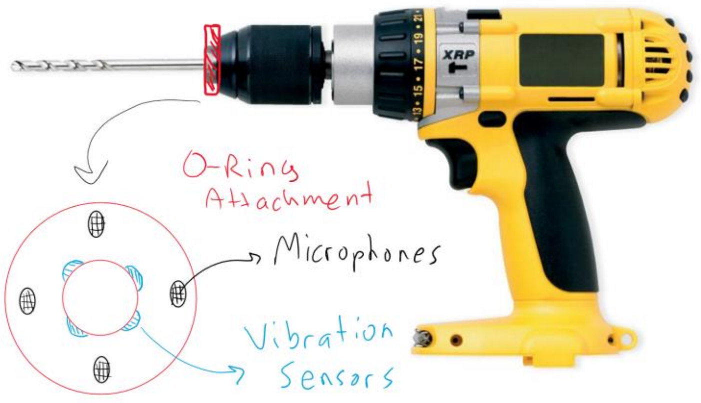


Figure 4

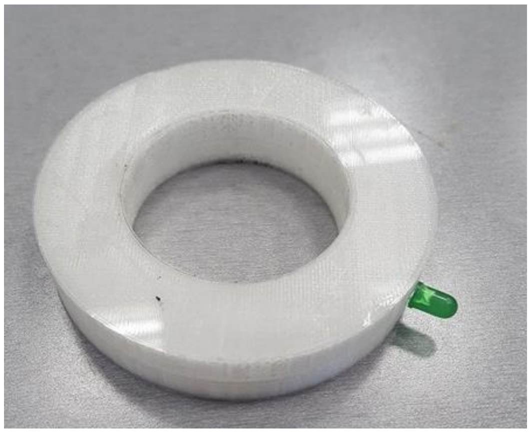


Figure 5

Figure 5 shows our MVP that we showed to our potential customers. Most people thought the position of the O-ring was good, but had concerns about clearance, durability, and blocking the view of the hole that's being drilled. In addition, people said that they would prefer if the device have rechargeable batteries, and that our product seemed viable and that they would buy it.

Our technology demonstration at the end of the semester was to show the ability to take live fast fourier transforms in real time. This hurdle would be the first step in developing the technology behind our project because it would be the main method of processing data from our sensors. Figures 6 and 7 below show the screen shots of our live fourier transform in action.

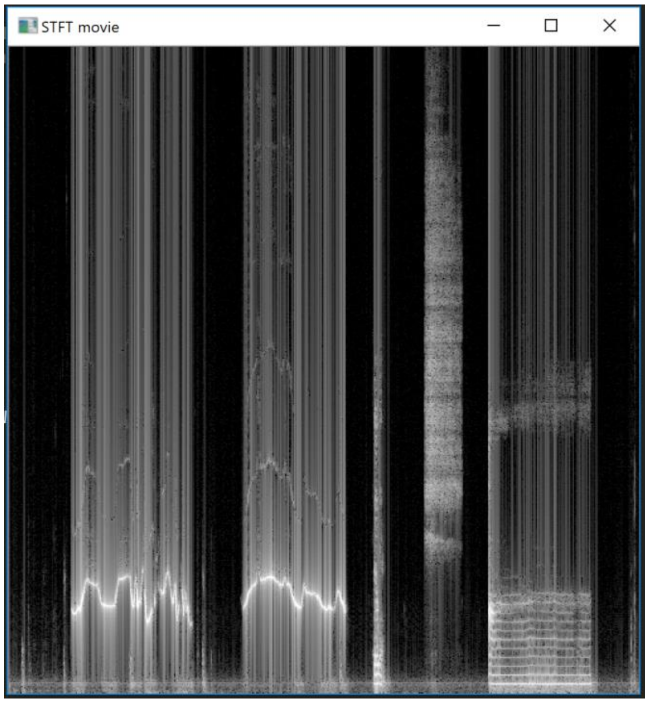


Figure 6

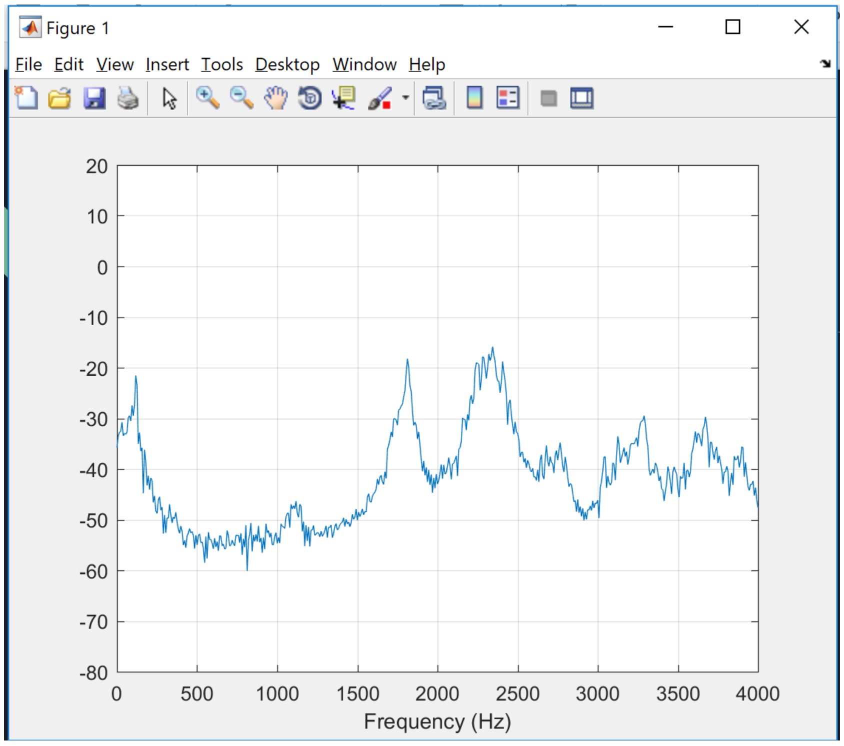


Figure 7

Shortly after our technology demonstration we attempted to test our theories about material frequency profiles with the vibration sensors and microphones that we bought. We found very little success when we tried to process the data. Upon further research we found the problem of material prediction behind walls was research being done by PhD students using RF reflections. We decided that the smart drill was too complex to implement so we decided to switch over to our current idea of the smart drip device.

Prototype Development, Technical Discussion, and Demonstration

Due to the fact that we switched our idea, our new problem to solve is to help busy nurses keep better track of their patients when they get busy. We want to centralize data for IV drip lines for all patients on a GUI that a nurse can look at and be able to monitor a patient's progress. Our new value proposition is:

"We present a small add on device to any IV setup that securely monitors volume and flow rate which is wirelessly displayed on a GUI."

In addition, our new target customers are hospitals and veterinary clinics. Our desired features for this device are to have an IR beam sensor, a microcontroller, a wireless link, and a GUI with real-time graphs. Figure 8 below shows the very basic system overview of how our product works. The drip device would use an IR sensor placed on the drip chamber that would count the number of drops, crunch the volume and the flow rate, and send the data over WIFI to a GUI.

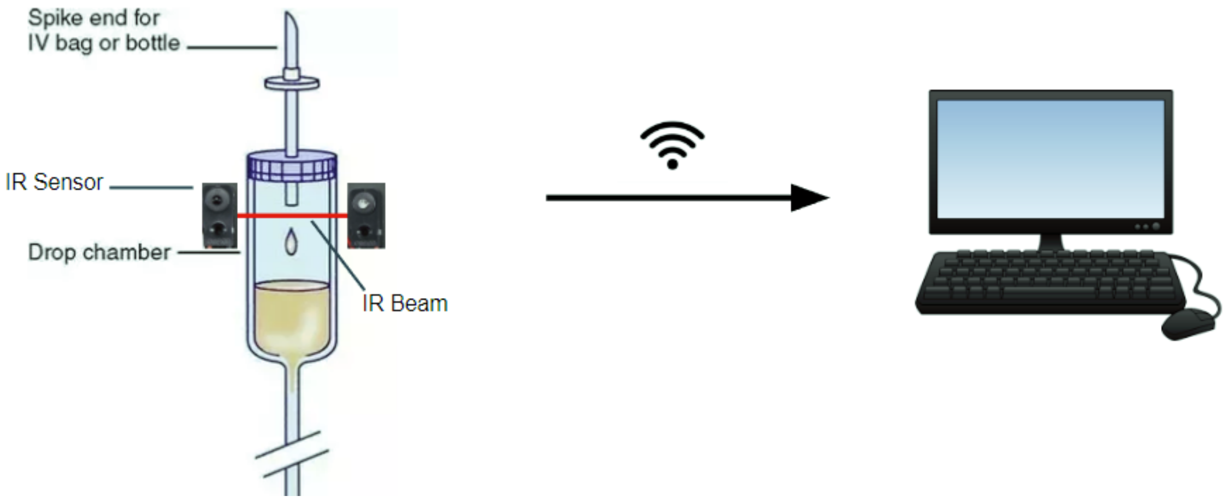
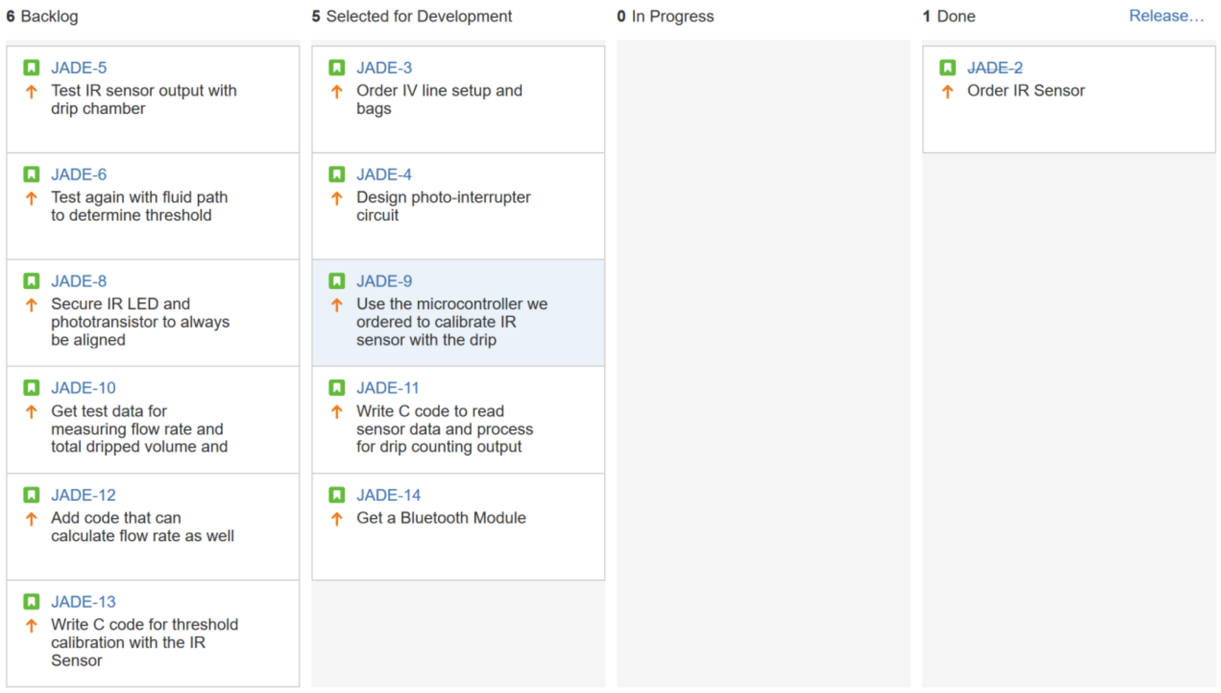


Figure 8

Prototype Development Schedule

Due to the fact that we changed our idea we had to do very quickly do testing to ensure that the technology that we chose works for our product. For the first week on the JIRA board we had a lot of things selected for development involving the purchasing of the sensors and microcontrollers, as well as design the necessary circuits to be able to detect droplets due to the translucent property of water.



6 Backlog

- JADE-5 Test IR sensor output with drip chamber
- JADE-6 Test again with fluid path to determine threshold
- JADE-8 Secure IR LED and phototransistor to always be aligned
- JADE-10 Get test data for measuring flow rate and total dripped volume and
- JADE-12 Add code that can calculate flow rate as well
- JADE-13 Write C code for threshold calibration with the IR Sensor

5 Selected for Development

- JADE-3 Order IV line setup and bags
- JADE-4 Design photo-interrupter circuit
- JADE-9 Use the microcontroller we ordered to calibrate IR sensor with the drip
- JADE-11 Write C code to read sensor data and process for drip counting output
- JADE-14 Get a Bluetooth Module

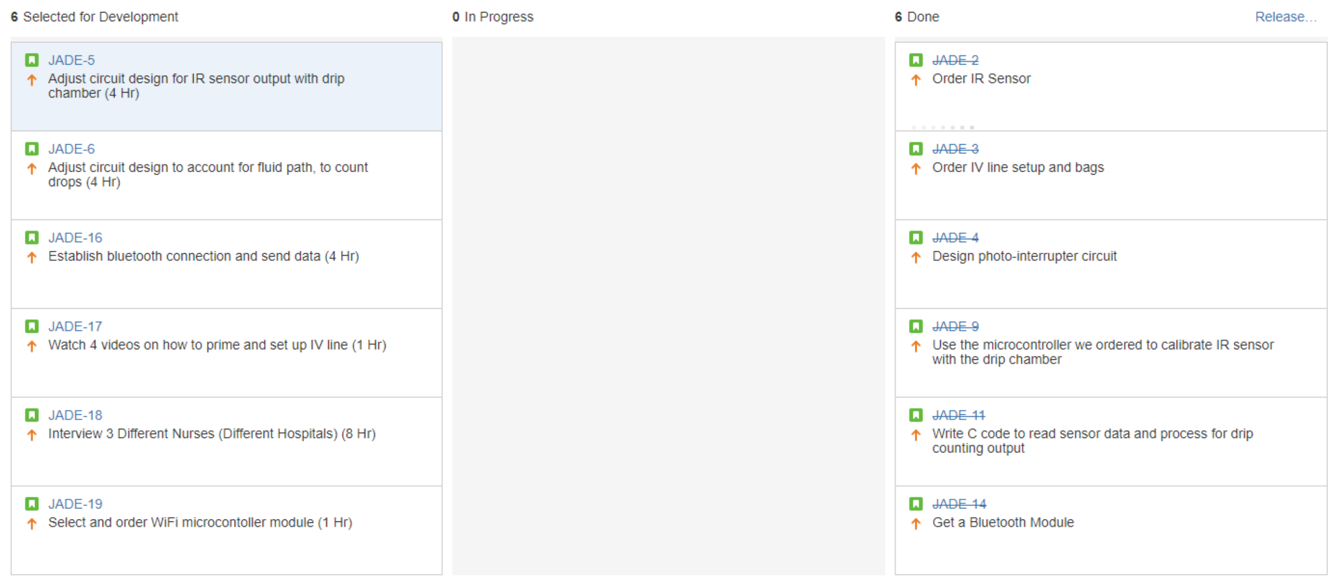
0 In Progress

1 Done Release...

- JADE-2 Order IR Sensor

Figure 9: Week 1 JIRA Board

For the second week, we learned how to prep an IV line, how to establish a Bluetooth connection, basic customer discovery interviewing nurses, and ordering a few other things. We originally proposed to have our device use a Bluetooth link instead of a WIFI link because of the simplicity of Bluetooth over the complex protocols of WIFI.



6 Selected for Development	0 In Progress	6 Done Release...
JADE-5 Adjust circuit design for IR sensor output with drip chamber (4 Hr)		~~JADE-2~~ Order IR Sensor
JADE-6 Adjust circuit design to account for fluid path, to count drops (4 Hr)		~~JADE-3~~ Order IV line setup and bags
JADE-16 Establish bluetooth connection and send data (4 Hr)		~~JADE-4~~ Design photo-interrupter circuit
JADE-17 Watch 4 videos on how to prime and set up IV line (1 Hr)		~~JADE-9~~ Use the microcontroller we ordered to calibrate IR sensor with the drip chamber
JADE-18 Interview 3 Different Nurses (Different Hospitals) (8 Hr)		~~JADE-11~~ Write C code to read sensor data and process for drip counting output
JADE-19 Select and order WiFi microcontoller module (1 Hr)		~~JADE-14~~ Get a Bluetooth Module

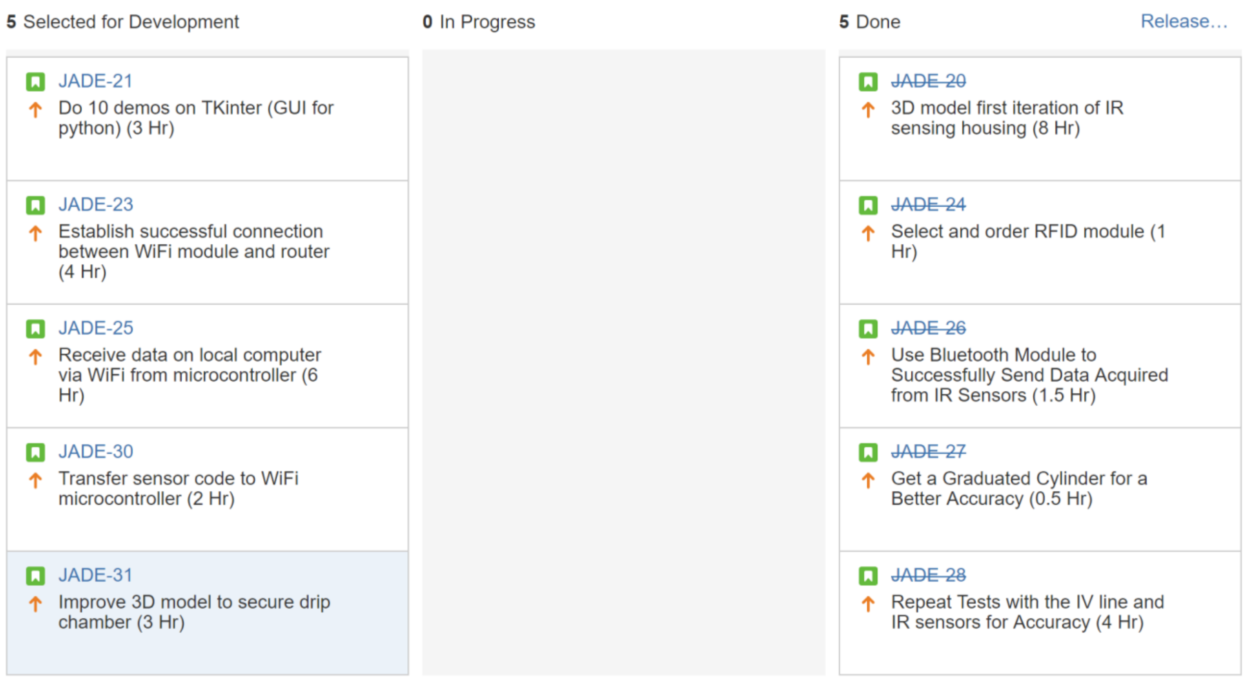
Figure 10: Week 2 JIRA Board

When we conducted the interviews with nurses we found that the nurses would like RFID tags on saline bags and patients to identify saline concentration and patient name on the GUI. In addition, they wanted control features from the GUI to the device such as pause, resume, and new drip options because medicine is often given in a separate bag in addition to the saline. Overall, the nurses that we interviewed seem to like our product and the concept of centralized data was very appealing.

For the third week, we want to get test data from our sensors to compare with actual volume measurements. In addition, we also started designing the housing for the IR sensors and learning how to work with SolidWorks to get this task done.

For the fourth week, we wanted to get a 3D model of our sensor housing, we wanted to successfully use Bluetooth to send and acquire data on a mobile device, and we wanted to repeat the tests we did in the prior week to get more data on how accurate our device is by using a graduated cylinder.

For the fifth week, we decided to pivot away from working with Bluetooth because we realized that it would be very difficult to transfer code written for Bluetooth to WIFI due to their vastly different protocols. We decided to research and work with the WIFI module that we had previously bought to interface with our device. We also wanted to get started with work on our GUI and the frontend code that needed to be completed for that.



5 Selected for Development	0 In Progress	5 Done Release…
JADE-21 Do 10 demos on TKinter (GUI for python) (3 Hr)		~~JADE-20~~ 3D model first iteration of IR sensing housing (8 Hr)
JADE-23 Establish successful connection between WiFi module and router (4 Hr)		~~JADE-24~~ Select and order RFID module (1 Hr)
JADE-25 Receive data on local computer via WiFi from microcontroller (6 Hr)		~~JADE-26~~ Use Bluetooth Module to Successfully Send Data Acquired from IR Sensors (1.5 Hr)
JADE-30 Transfer sensor code to WiFi microcontroller (2 Hr)		~~JADE-27~~ Get a Graduated Cylinder for a Better Accuracy (0.5 Hr)
JADE-31 Improve 3D model to secure drip chamber (3 Hr)		~~JADE-28~~ Repeat Tests with the IV line and IR sensors for Accuracy (4 Hr)

Figure 11: Week 5 JIRA Board

For sixth week, we had printed our first iteration of the sensor housing and we needed to test it with our sensors, we also wanted to be able to read RFID tags with the module we bought, and we also wanted to create a real-time graph in a GUI.

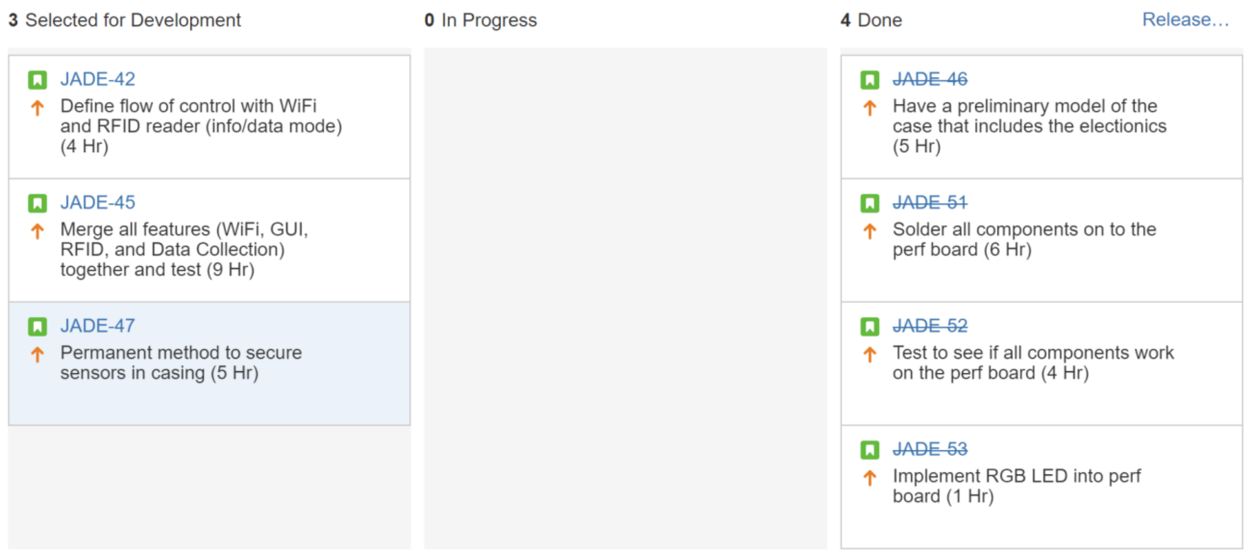
For the seventh week, we wanted to be merge all of the code that we have written for RFID, the WIFI module, and the basic sensor data acquisition code. In addition, we wanted to update our 3D model to make it slimmer.

For the eighth week, we wanted to make our GUI and server run concurrently and be able to pull data and refresh the GUI, and to rewrite the server code so that it is able to handle multiple simultaneous client connections. To be able to gather data from numerous clients we needed to program our own communication protocol.

For the ninth week, we wanted to make our GUI update multiple graphs in real time concurrently with each patient connected. We also wanted our GUI to be able to display multiple patients, we also wanted to order a rechargeable battery for our system, and we began ordering parts for our second client.

For the tenth week, we wanted to have a preliminary model for a new case that can hold the sensors and the electronics such as WIFI module, the microcontroller, LED, and RFID. We also wanted to solder all of our electrical components onto a perf board then test each component to ensure it is still working correctly. Finally, we wanted to have an indication light on the board that will let the user know about the status of the device.

For the final week, we just needed to finish up defining the flow of control in our code between data mode and setup mode. We also needed to merge all of our written code together and test it. Finally, we needed to find a way to permanently secure the the sensors.



3 Selected for Development	0 In Progress	4 Done
JADE-42 Define flow of control with WiFi and RFID reader (info/data mode) (4 Hr)		~~JADE-46~~ Have a preliminary model of the case that includes the electionics (5 Hr)
JADE-45 Merge all features (WiFi, GUI, RFID, and Data Collection) together and test (9 Hr)		~~JADE-51~~ Solder all components on to the perf board (6 Hr)
JADE-47 Permanent method to secure sensors in casing (5 Hr)		~~JADE-52~~ Test to see if all components work on the perf board (4 Hr)
		~~JADE-53~~ Implement RGB LED into perf board (1 Hr)

Release…

Figure 12: Final Week JIRA Board

Technical Discussion

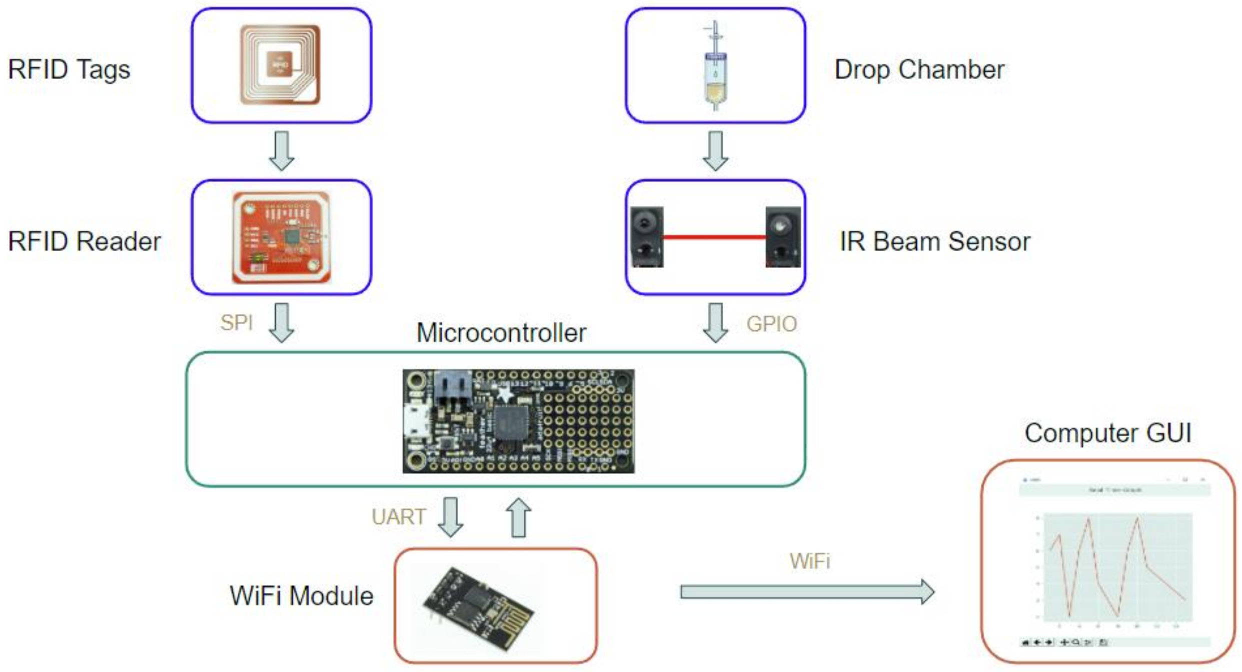


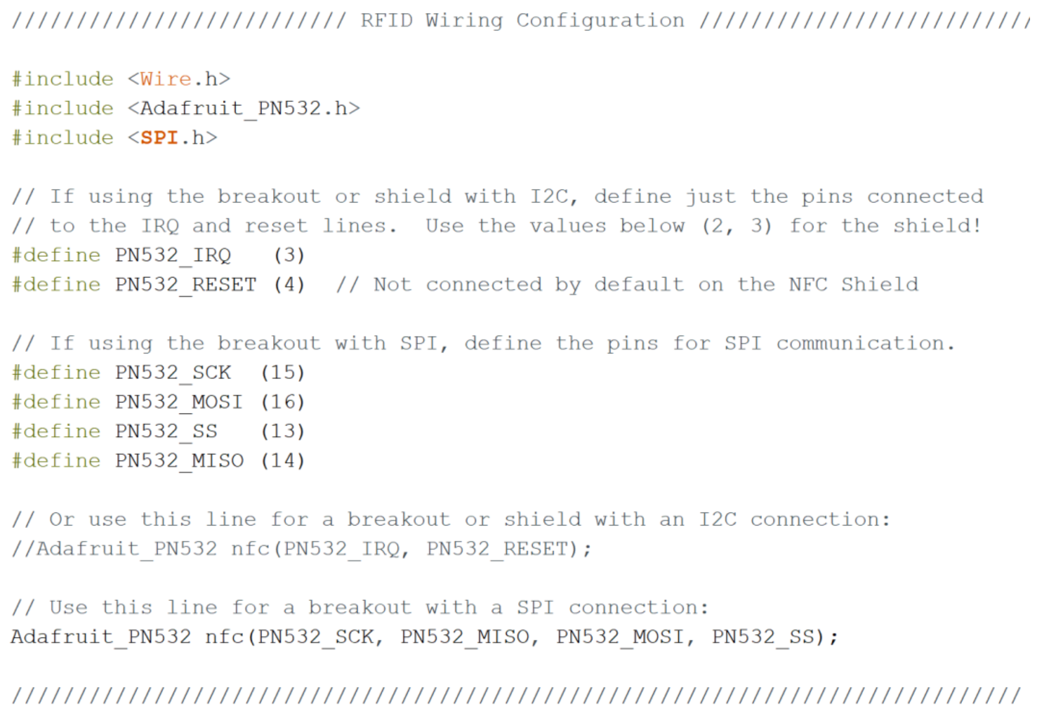
Figure 13

Figure 13 shows a more in-depth view of how each component in our system communicated with the microcontroller and the GUI. The RFID reader and the IR beam sensor are collecting data from the external world and sending the data to our microcontroller, the Adafruit Feather Proto Board. The RFID reader sends the data via serial SPI and the data collection is done via an external interrupt trigger that increments a counter. The raw drop count is then converted to a volume value using the drop factor and the volume flow rate is calculated between each drop calculation. The data is then sent over a serial UART line to our ESP-8266-01 WIFI module which transmits the data over WIFI to our GUI.

Microcontroller Code Breakdown

The following microcontroller code is written for any Adafruit 32u4 Processor. The code goes through several different sequential steps. First, there are two “main” functions when coding in Arduino IDE. There is a setup function and a loop function. The loop is essentially an infinite while loop.

To begin, the RFID reader pin configuration is set as shown below. It can be communicated with multiple interfaces. SPI was chosen since the pins were available, and SPI is a faster protocol than I2C.



```
///////////////////////// RFID Wiring Configuration /////////////////////////

#include <Wire.h>
#include <Adafruit_PN532.h>
#include <SPI.h>

// If using the breakout or shield with I2C, define just the pins connected
// to the IRQ and reset lines.  Use the values below (2, 3) for the shield!
#define PN532_IRQ   (3)
#define PN532_RESET (4)  // Not connected by default on the NFC Shield

// If using the breakout with SPI, define the pins for SPI communication.
#define PN532_SCK  (15)
#define PN532_MOSI (16)
#define PN532_SS   (13)
#define PN532_MISO (14)

// Or use this line for a breakout or shield with an I2C connection:
//Adafruit_PN532 nfc(PN532_IRQ, PN532_RESET);

// Use this line for a breakout with a SPI connection:
Adafruit_PN532 nfc(PN532_SCK, PN532_MISO, PN532_MOSI, PN532_SS);

////////////////////////////////////////////////////////////////////////////////
```

The function lenHelper was used to help determine the WIFI buffer size that was needed to each data transmission. Our data consisted of float values that could vary in length. LenHelper will be able to quickly compute how many characters are in a certain floating point number. For example, 10.21 would return 5 for the number of characters. The checkServerMsg function polls for a response from the server. This usually occurs when the microcontroller sends some piece of data to the server and is waiting for an acknowledgement that the data was received.

```
int lenHelper(float x)
{
    if(x>=1000000000) return 13;
    if(x>=100000000) return 12;
    if(x>=10000000) return 11;
    if(x>=1000000) return 10;
    if(x>=100000) return 9;
    if(x>=10000) return 8;
    if(x>=1000) return 7;
    if(x>=100) return 6;
    if(x>=10) return 5;
    return 4;
}

void checkServerMsg(String check)
{
  if(Serial1.available()>0)
  {
    while(Serial1.available()>0)
    {
      String msg = Serial1.readStringUntil('\n');
      Serial.println(msg);
      if(msg == check)
      {
        break;
      }
      break;
     }
  }
}
```

Several global variables were also set, such that they can be manipulated in any function and by any peripheral chosen. Each variable's purpose is explained in the code comments.

```
// Sensor Variables
volatile unsigned int counter;        // Drip count
volatile unsigned long t = 0;         // Time of current drip
volatile float t_diff = 0;            // Time between current drip and previous
float x;                      // Total code elapsed time
unsigned short stop_time = 10000;     // Threshold time bewteen drops to determine when IV bag is finished
float vol = 0, prevvol = 0;           // Current and previous volume
float flowrate = 0;                   // Flowrate of IV line
float drop_factor;                    // IV line drop factor
float timeCount = 0;
int BuffSizeData = 0;
int BuffSizeEnd = 0;

// RFID Variables
boolean success;                                  // Store whether scan was successful or not
uint8_t uid_patient[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned patient UID
uint8_t uid_ivbag[] = { 0, 0, 0, 0, 0, 0, 0 };    // Buffer to store the returned IV bag UID
uint8_t uidLength;                                // Length of the UID (4 or 7 bytes depending on ISO14443A card type)

uint8_t patient_tag[] = { 192, 79, 109, 163, 0, 0, 0 };  // Expected patient tag
uint8_t ivbag_tag[] = { 22, 216, 219, 217, 0, 0, 0 };    // Expected IV bag tag
boolean patient_success = true;                          // State of whether or not correct tag was scanned for patient info
boolean ivbag_success = true;                            // State of whether or not correct tag was scanned for IV bag info

String patientName;
```

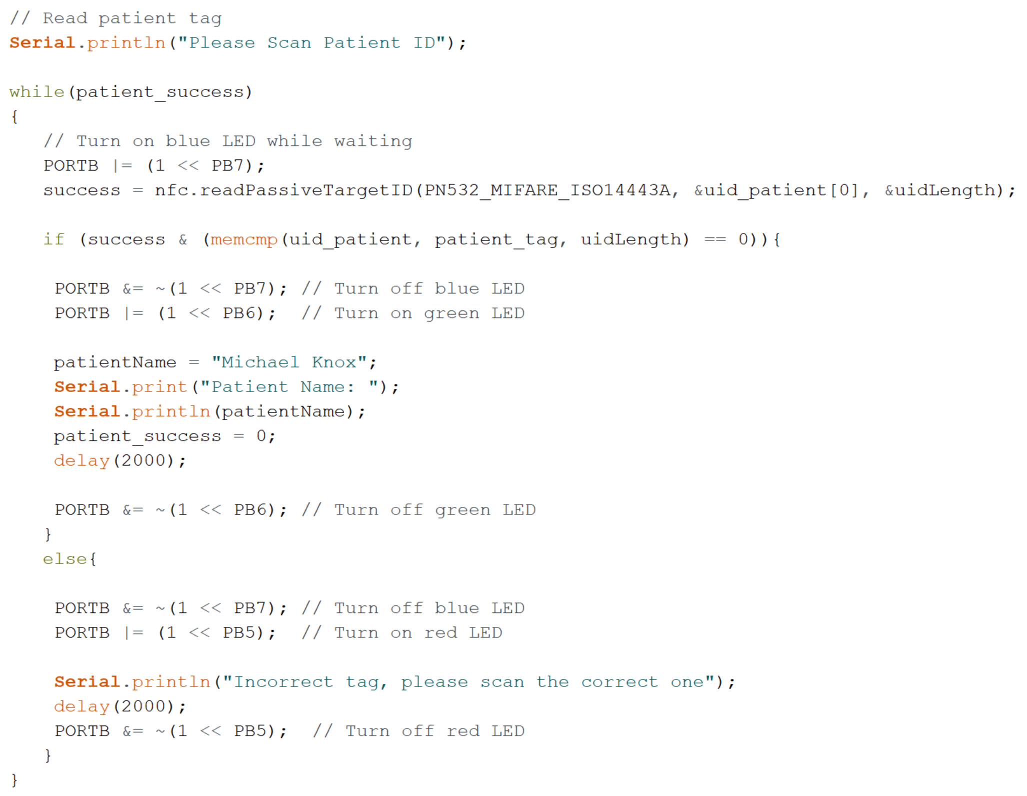
Here the setup function as mentioned above is declared, where one-time setup parameters are configured. First the RGB LED is configured by setting certain bits in the GPIO registers.

```
void setup() {

  // PIN SETUP
  DDRD &= 0b11111110;  // Make sure PD0 is input for sensor (which is INT0 Pin 3)
  PORTD |= 0b00000001; // Turn on pullup resistor for PD0 (THIS MAY NOT BE NEEDED IF EXTERNAL PULLUP IS USED)
  DDRB |= 0b11100000;  // Make PB5,6,7 output for RGB LED. RED -> Pin 9, Green -> Pin 10, Blue -> Pin 11

  // EXTERNAL INTERRUPT
  EICRA = 0b00000010;  // Faliing edge of INT0 will generate interrupt request
  EIMSK = 0b00000001;  // Enable external pin interrupt for INT0
  // EIFR is ths flag register for external interrupts

  Serial.begin(115200);
```

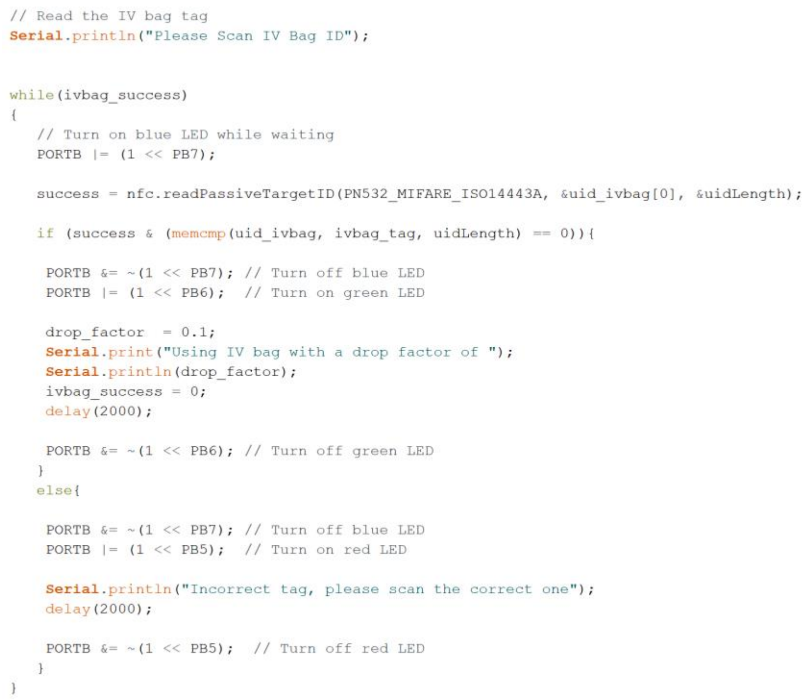
Then the RFID module is set up by turning on the NFC, and debugging the device itself (checks to see if the board is present or found).

```
///////////////////////// RFID SETUP /////////////////////////

  nfc.begin();

  uint32_t versiondata = nfc.getFirmwareVersion();
  if (! versiondata) {
    Serial.print("Didn't find PN53x board");
    while (1); // halt
  }

  // Got ok data, print it out!
  Serial.print("Found chip PN5"); Serial.println((versiondata>>24) & 0xFF, HEX);
  Serial.print("Firmware ver. "); Serial.print((versiondata>>16) & 0xFF, DEC);
  Serial.print('.'); Serial.println((versiondata>>8) & 0xFF, DEC);

  // Set the max number of retry attempts to read from a card
  // This prevents us from waiting forever for a card, which is
  // the default behaviour of the PN532.
  nfc.setPassiveActivationRetries(0xFF);

  // configure board to read RFID tags
  nfc.SAMConfig();
```

Once the RFID module is found, tags can be scanned and information can be obtained. In the following code, the patient and IV bag tags are scanned. A blue LED means waiting for a tag, green is correct, and red is incorrect.



```
// Read patient tag
Serial.println("Please Scan Patient ID");

while(patient_success)
{
   // Turn on blue LED while waiting
   PORTB |= (1 << PB7);
   success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid_patient[0], &uidLength);

   if (success & (memcmp(uid_patient, patient_tag, uidLength) == 0)){

    PORTB &= ~(1 << PB7); // Turn off blue LED
    PORTB |= (1 << PB6);  // Turn on green LED

    patientName = "Michael Knox";
    Serial.print("Patient Name: ");
    Serial.println(patientName);
    patient_success = 0;
    delay(2000);

    PORTB &= ~(1 << PB6); // Turn off green LED
   }
   else{

    PORTB &= ~(1 << PB7); // Turn off blue LED
    PORTB |= (1 << PB5);  // Turn on red LED

    Serial.println("Incorrect tag, please scan the correct one");
    delay(2000);
    PORTB &= ~(1 << PB5);  // Turn off red LED
   }
}
```



```
// Read the IV bag tag
Serial.println("Please Scan IV Bag ID");


while(ivbag_success)
{
   // Turn on blue LED while waiting
   PORTB |= (1 << PB7);

   success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid_ivbag[0], &uidLength);

   if (success & (memcmp(uid_ivbag, ivbag_tag, uidLength) == 0)){

    PORTB &= ~(1 << PB7); // Turn off blue LED
    PORTB |= (1 << PB6);  // Turn on green LED

    drop_factor  = 0.1;
    Serial.print("Using IV bag with a drop factor of ");
    Serial.println(drop_factor);
    ivbag_success = 0;
    delay(2000);

    PORTB &= ~(1 << PB6); // Turn off green LED
   }
   else{

    PORTB &= ~(1 << PB7); // Turn off blue LED
    PORTB |= (1 << PB5);  // Turn on red LED

    Serial.println("Incorrect tag, please scan the correct one");
    delay(2000);

    PORTB &= ~(1 << PB5);  // Turn off red LED
   }
}
```

The last part in this section sets up the WIFI module and communication. The microcontroller sets up a TCP connection to our server and waits for acknowledgement for a password then for a name. Once connected and the setup information is sent, the LED turns purple to demonstrate the device is in Data mode.

```
  ////////////////////////// WiFi SETUP //////////////////////////

  ///Initialize WiFi////
  Serial1.begin(9600);
  //Serial1.println(String("AT+RST"));
  //Serial1.println(String("AT+CWMODE=1"));
  //Serial1.println(String("AT+CWJAP=\"Abraham Linksys\",\"dziedzic"));
  delay(1000);
  Serial1.println(String("AT+CIPSTART=\"TCP\",\"192.168.1.108\",8888"));
  delay(1000);
  checkServerMsg("Pass?\n");
  delay(1000);
  Serial1.println("AT+CIPSEND=" + String("3"));
  delay(1000);
  Serial1.print("abc");
  delay(1000);
  checkServerMsg("Name?\n");
  delay(1000);
  String patientNameLen = String(patientName.length());
  Serial1.println("AT+CIPSEND=" + String(patientNameLen));
  delay(1000);
  Serial1.print(patientName);

  // Purple light to demonstrate DATA mode
  PORTB |= (1 << PB5);  // Turn on red LED
  PORTB |= (1 << PB7);  // Turn on blue LED
}
```

Next, the loop function is what iterates over and over when the microcontroller is on. Essentially, it checks what the current drip count is and calculates the volume using the drop factor. In addition, the flowrate is also calculated using the difference of the two most recent timestamps of drops. These values are also transmitted over WIFI only once there is a change in volume.

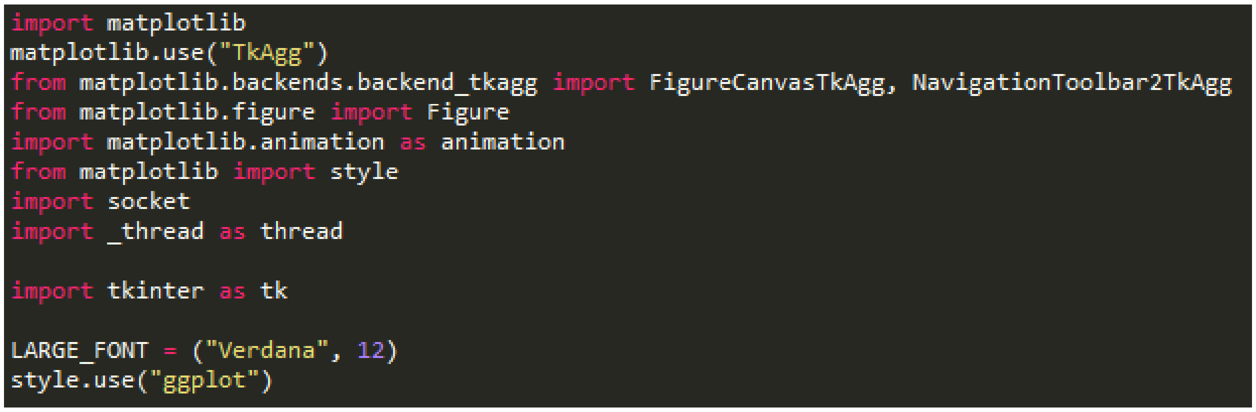
```
void loop() {
  // Calculated volume
  vol = drop_factor * counter;

  // Calculated flowrate
  flowrate = (drop_factor / (t_diff/1000)) * 60;

  /////WiFi Buffer Setup/////
  BuffSizeEnd = 24+lenHelper(vol);

  // Print results
  checkServerMsg("OK\n");
  if (vol == 0.1)
  {
    x = ((float)millis()/60000.0);
  }
  if (prevvol != vol)
  {
    timeCount = ((float)millis()/60000.0) - x;
    Serial.print(timeCount,6);
    Serial.print(",");
    Serial.print(vol, 2);
    Serial.print(",");
    Serial.println(flowrate, 2);

    BuffSizeData = lenHelper(timeCount)+ 4 + lenHelper(vol) + lenHelper(flowrate) + 2;
    Serial1.println("AT+CIPSEND=" + String(BuffSizeData));
    delay(500);
    Serial1.print(timeCount,6);
    Serial1.print(",");
    Serial1.print(vol,2);
    Serial1.print(",");
    Serial1.print(flowrate,2);
   }
  // Update previous volume value
  prevvol = vol;
}
```

Finally, an interrupt function is used to capture the drops. It triggers on an edge (when a drop breaks the IR beam), and then increments the counter. There is also a debouncing condition set to prevent an inaccurate excess of drips counted.

```
// Interrupt functions
ISR(INT0_vect) {

  if (millis() - t > 90){    // Debouncing
    t_diff = millis() - t;   // Time differnece from previous drip
    counter++;               // Increment drip counter
    t = millis();            // Time of current drip
  }

 print_state = true;         // Allow final volume printing again
}
```

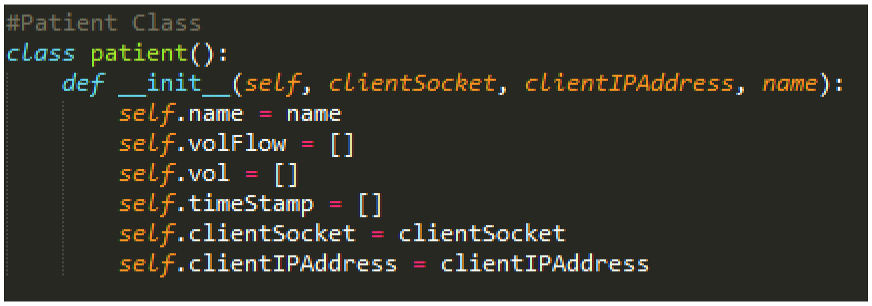
Python Server and GUI Code Breakdown

Our server backend and GUI was programmed in Python 3. The snippet of code below shows all of the libraries that we used in order to make everything work. The matplot libraries were used to make all of the graphs, the socket library was used to create the server, the tkinter library was used for making the GUI, and the thread library was used to run the GUI and the server on parallel threads to have them run concurrently.



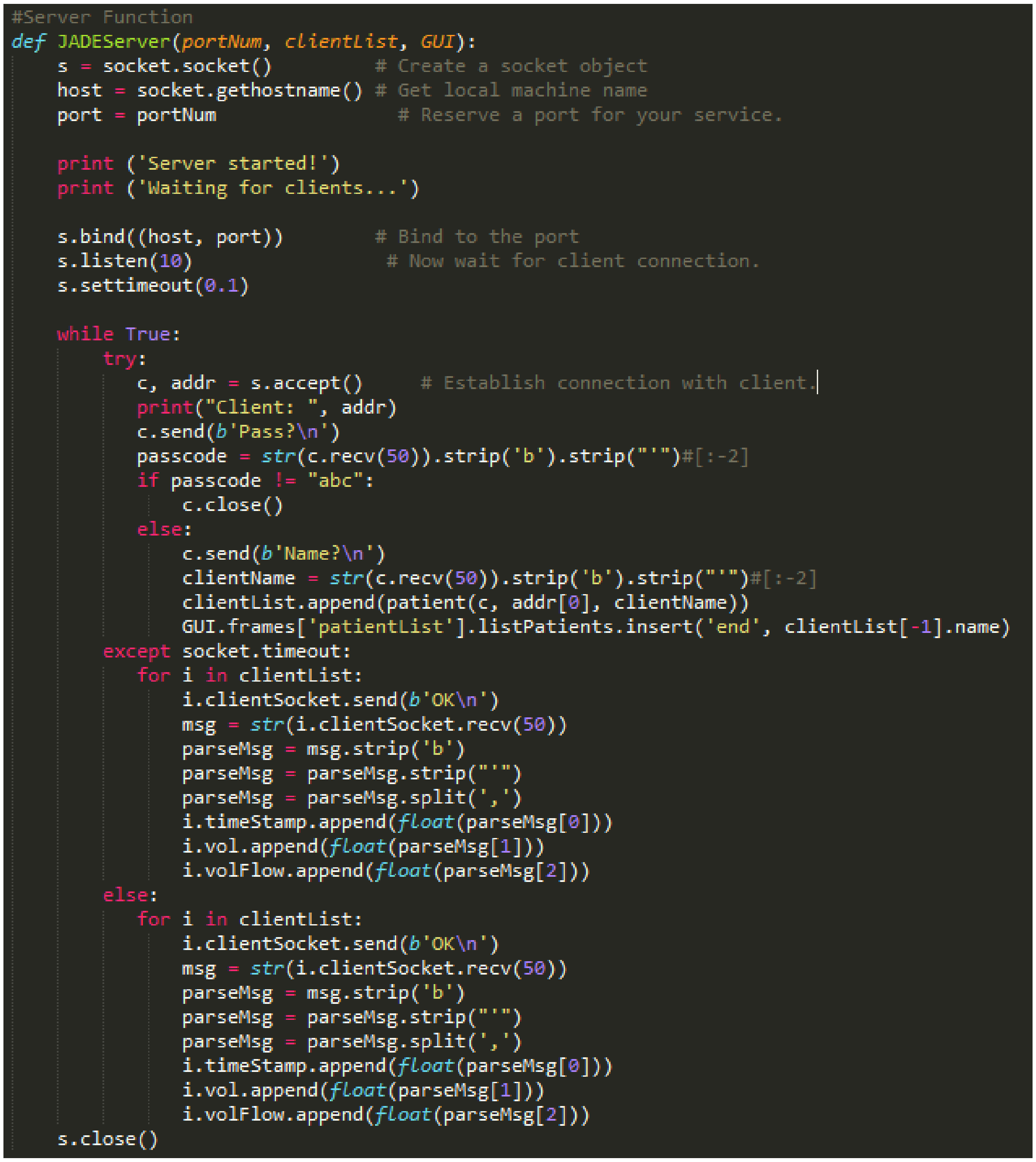
```python
import matplotlib
matplotlib.use("TkAgg")
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2TkAgg
from matplotlib.figure import Figure
import matplotlib.animation as animation
from matplotlib import style
import socket
import _thread as thread

import tkinter as tk

LARGE_FONT = ("Verdana", 12)
style.use("ggplot")
```

The code below is the definition for the patient class that includes the patient’s name and their data that would be sent from the microcontroller.



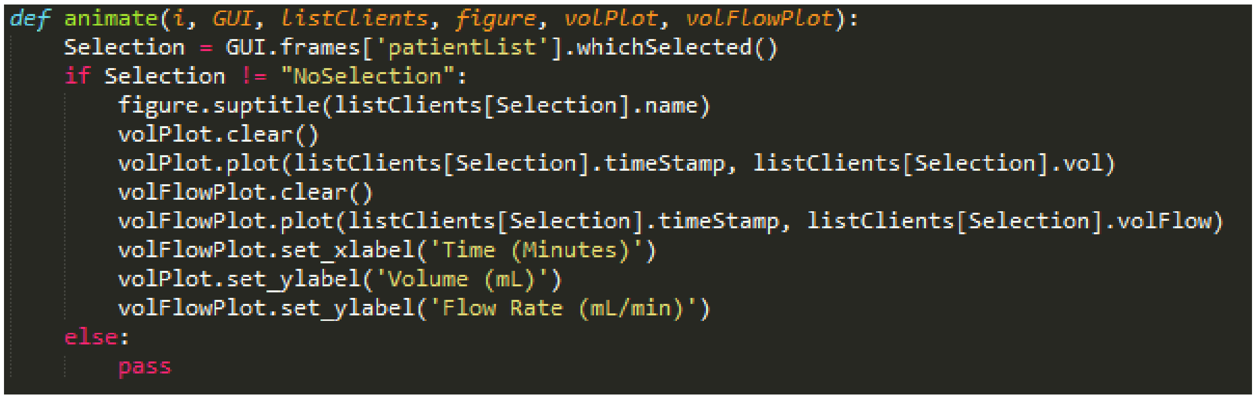
```python
#Patient Class
class patient():
    def __init__(self, clientSocket, clientIPAddress, name):
        self.name = name
        self.volFlow = []
        self.vol = []
        self.timeStamp = []
        self.clientSocket = clientSocket
        self.clientIPAddress = clientIPAddress
```

The JADEServer is the function that contains our entire server. A socket is first created and attached to a port to listen to on the computer. The server then sits and polls for new clients to connect. Everytime a client connects, the server asks for a password and if the password matches then the client is accepted and a new patient object is created for the given IP address that connected. The for loops inside the while loop iterate through a list of connected clients and polls the client for data by sending a string OK to the client. The data is then received from the client and parsed into the respective array for each patient, timeStamp, vol, and volFlow.



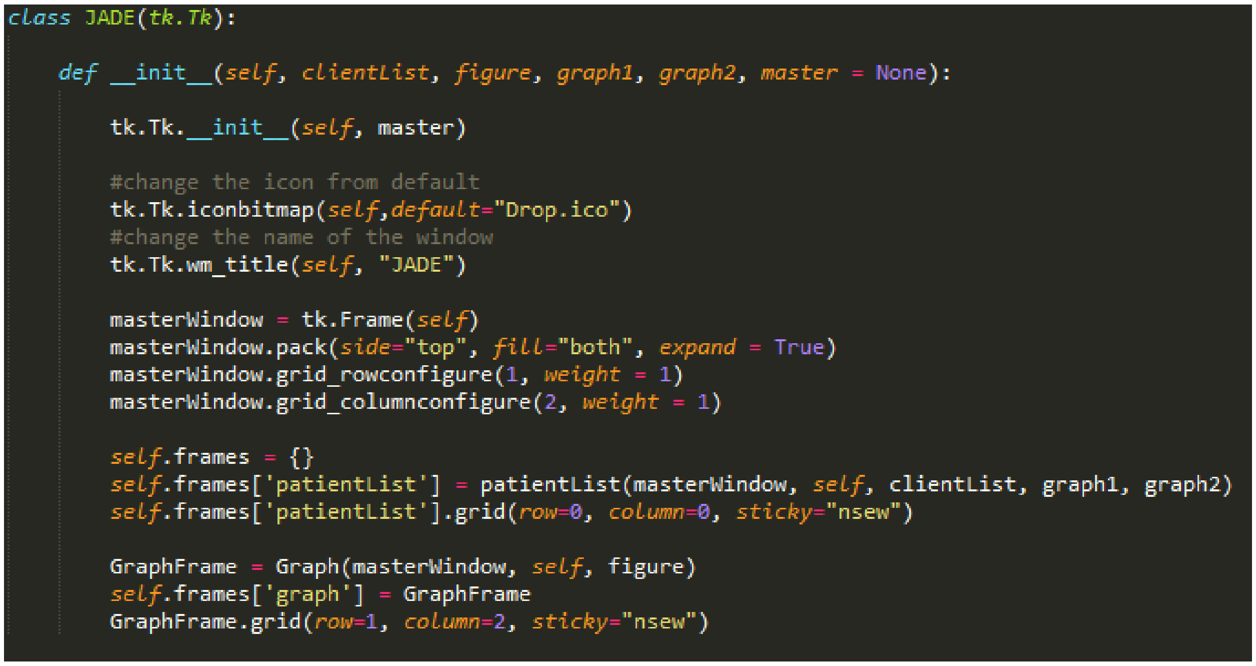
```
#Server Function
def JADEServer(portNum, clientList, GUI):
    s = socket.socket()         # Create a socket object
    host = socket.gethostname() # Get local machine name
    port = portNum               # Reserve a port for your service.

    print ('Server started!')
    print ('Waiting for clients...')

    s.bind((host, port))        # Bind to the port
    s.listen(10)                 # Now wait for client connection.
    s.settimeout(0.1)

    while True:
        try:
           c, addr = s.accept()     # Establish connection with client.
           print("Client: ", addr)
           c.send(b'Pass?\n')
           passcode = str(c.recv(50)).strip('b').strip("'")#[:-2]
           if passcode != "abc":
               c.close()
           else:
               c.send(b'Name?\n')
               clientName = str(c.recv(50)).strip('b').strip("'")#[:-2]
               clientList.append(patient(c, addr[0], clientName))
               GUI.frames['patientList'].listPatients.insert('end', clientList[-1].name)
        except socket.timeout:
           for i in clientList:
               i.clientSocket.send(b'OK\n')
               msg = str(i.clientSocket.recv(50))
               parseMsg = msg.strip('b')
               parseMsg = parseMsg.strip("'")
               parseMsg = parseMsg.split(',')
               i.timeStamp.append(float(parseMsg[0]))
               i.vol.append(float(parseMsg[1]))
               i.volFlow.append(float(parseMsg[2]))
        else:
           for i in clientList:
               i.clientSocket.send(b'OK\n')
               msg = str(i.clientSocket.recv(50))
               parseMsg = msg.strip('b')
               parseMsg = parseMsg.strip("'")
               parseMsg = parseMsg.split(',')
               i.timeStamp.append(float(parseMsg[0]))
               i.vol.append(float(parseMsg[1]))
               i.volFlow.append(float(parseMsg[2]))
    s.close()
```

The animate function is used to update the live graphs that are displayed on the GUI. The animate function is later used in the main loop by the FuncAnimation function. The animate function determines which patient has been selected on the list and then fetches that user’s data from the list of connected clients on the server and pulls the data and updates the graph.



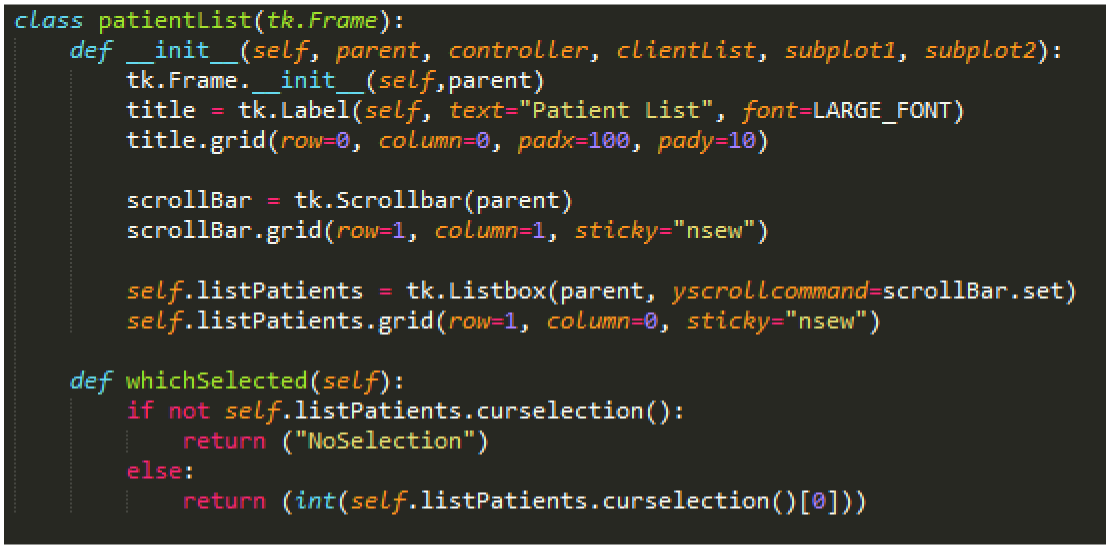
```
def animate(i, GUI, listClients, figure, volPlot, volFlowPlot):
    Selection = GUI.frames['patientList'].whichSelected()
    if Selection != "NoSelection":
        figure.suptitle(listClients[Selection].name)
        volPlot.clear()
        volPlot.plot(listClients[Selection].timeStamp, listClients[Selection].vol)
        volFlowPlot.clear()
        volFlowPlot.plot(listClients[Selection].timeStamp, listClients[Selection].volFlow)
        volFlowPlot.set_xlabel('Time (Minutes)')
        volPlot.set_ylabel('Volume (mL)')
        volFlowPlot.set_ylabel('Flow Rate (mL/min)')
    else:
        pass
```

The JADE class is the definition for our GUI and within it, Tk builds our master window along with all of the predefined frames inside of the master window.



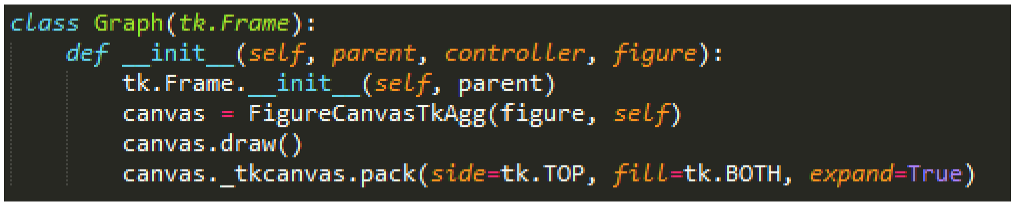
```
class JADE(tk.Tk):

    def __init__(self, clientList, figure, graph1, graph2, master = None):

        tk.Tk.__init__(self, master)

        #change the icon from default
        tk.Tk.iconbitmap(self,default="Drop.ico")
        #change the name of the window
        tk.Tk.wm_title(self, "JADE")

        masterWindow = tk.Frame(self)
        masterWindow.pack(side="top", fill="both", expand = True)
        masterWindow.grid_rowconfigure(1, weight = 1)
        masterWindow.grid_columnconfigure(2, weight = 1)

        self.frames = {}
        self.frames['patientList'] = patientList(masterWindow, self, clientList, graph1, graph2)
        self.frames['patientList'].grid(row=0, column=0, sticky="nsew")

        GraphFrame = Graph(masterWindow, self, figure)
        self.frames['graph'] = GraphFrame
        GraphFrame.grid(row=1, column=2, sticky="nsew")
```

The patientList class is the frame information for building the list of patients that is seen in the GUI on the left side. The function whichSelected determines which patient is selected on the list and returns its index in the list.



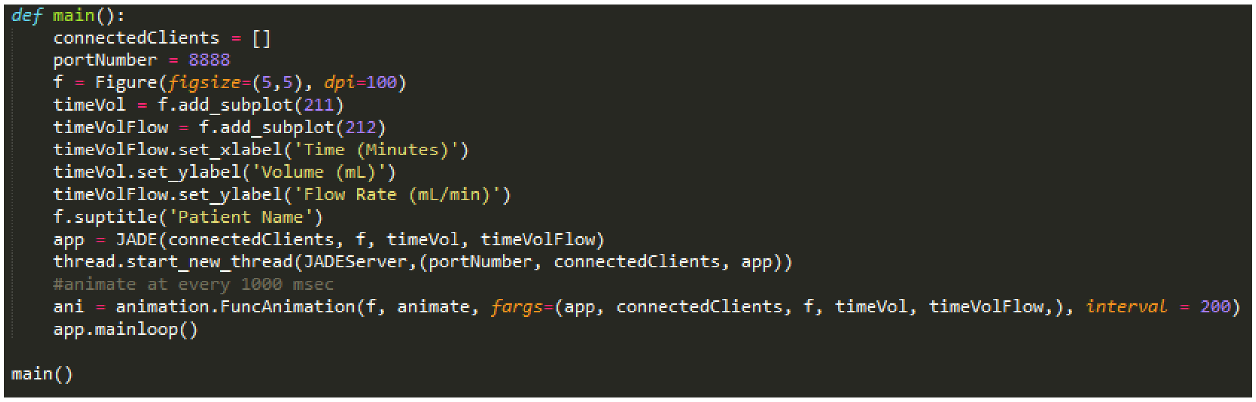
```
class patientList(tk.Frame):
    def __init__(self, parent, controller, clientList, subplot1, subplot2):
        tk.Frame.__init__(self,parent)
        title = tk.Label(self, text="Patient List", font=LARGE_FONT)
        title.grid(row=0, column=0, padx=100, pady=10)

        scrollBar = tk.Scrollbar(parent)
        scrollBar.grid(row=1, column=1, sticky="nsew")

        self.listPatients = tk.Listbox(parent, yscrollcommand=scrollBar.set)
        self.listPatients.grid(row=1, column=0, sticky="nsew")

    def whichSelected(self):
        if not self.listPatients.curselection():
            return ("NoSelection")
        else:
            return (int(self.listPatients.curselection()[0]))
```

The Graph class is the definition for graph frame that is created to the right of the patient list in the GUI. This class initializes the graph at the execution of the GUI.



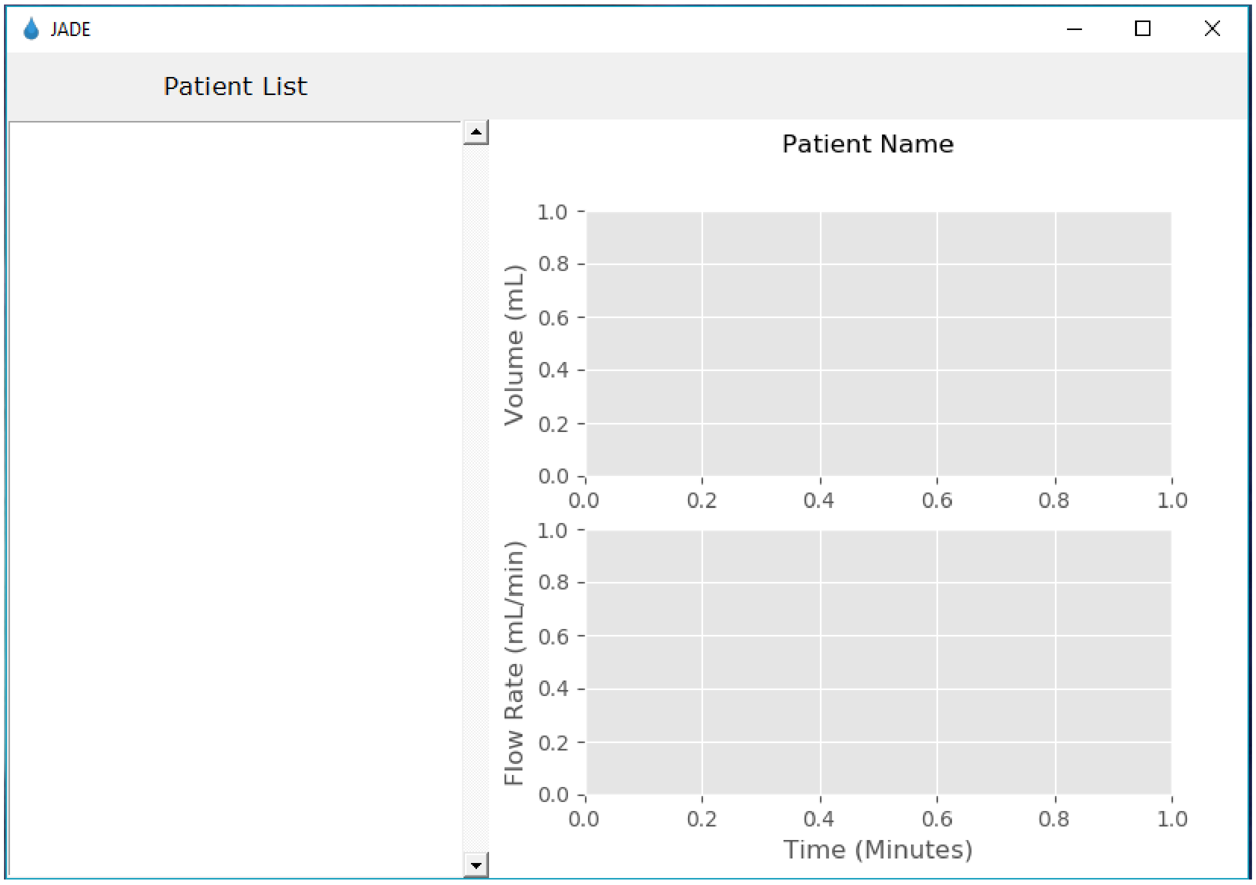
```
class Graph(tk.Frame):
    def __init__(self, parent, controller, figure):
        tk.Frame.__init__(self, parent)
        canvas = FigureCanvasTkAgg(figure, self)
        canvas.draw()
        canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=True)
```

The main function is the script that runs to setup all of the functions written above and starts the server and the GUI on seperate threads.

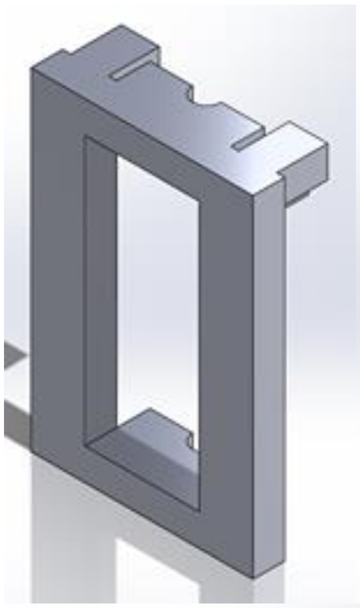
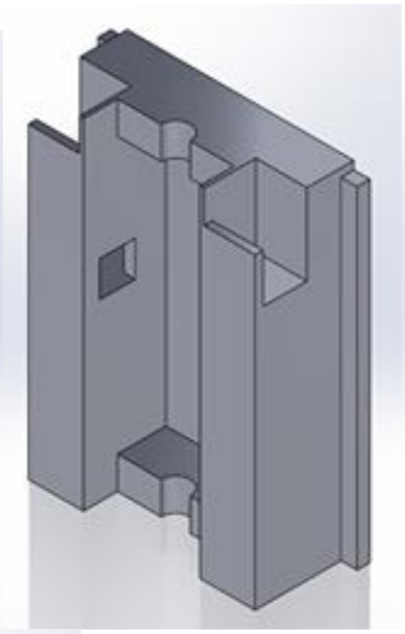
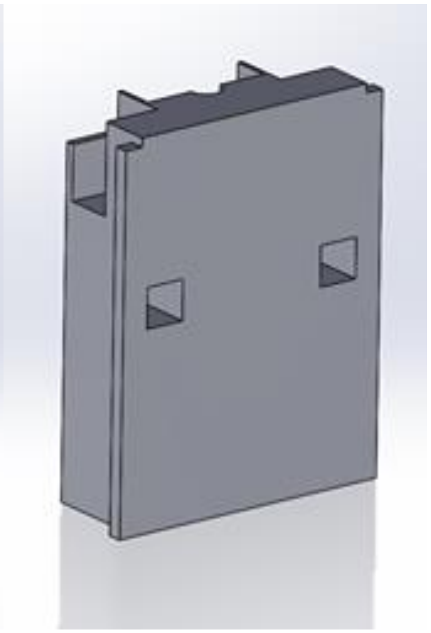
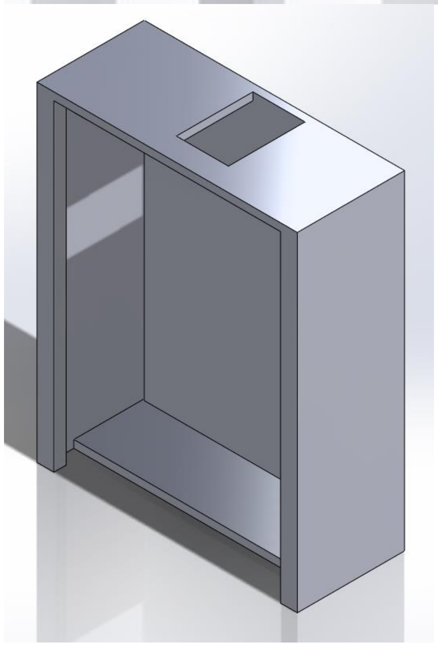


```
def main():
    connectedClients = []
    portNumber = 8888
    f = Figure(figsize=(5,5), dpi=100)
    timeVol = f.add_subplot(211)
    timeVolFlow = f.add_subplot(212)
    timeVolFlow.set_xlabel('Time (Minutes)')
    timeVol.set_ylabel('Volume (mL)')
    timeVolFlow.set_ylabel('Flow Rate (mL/min)')
    f.suptitle('Patient Name')
    app = JADE(connectedClients, f, timeVol, timeVolFlow)
    thread.start_new_thread(JADEServer,(portNumber, connectedClients, app))
    #animate at every 1000 msec
    ani = animation.FuncAnimation(f, animate, fargs=(app, connectedClients, f, timeVol, timeVolFlow,), interval = 200)
    app.mainloop()

main()
```

The picture below shows what our GUI looks like when all of the Python code above is executed. The left side is the list of all patients connected to the server and the right side is an empty graph that has been initialized.



3D Models

Appendix

ATMega32u4 (Adafruit Feather Proto) Microcontroller Datasheet

- http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf