Individual Lab Report - 1

Autonomous Reaming for Total Hip Replacement

HIPSTER | ARTHuR

Parker Hill

Team C:

Parker Hill | Kaushik Balasundar | Anthony Kyu

Sundaram Seivur | Gunjan Sethi

February 10th, 2022

Carnegie Mellon University
The Robotics Institute

Contents

1 Individual Progress

1.1 Sensor and Motors Lab

1.1.1 Ultrasonic Rangefinder

The LV-MaxSonar-EZ1 is an ultrasonic rangefinder which was relatively easy to interface with an arduino to detect objects from 6 inches away to 254 inches away with a resolution of approximately 1 inch. The rangefinder is powered by a 5V power supply and has two separate ways of outputting range data to an arduino, analog output and pulse width output. For simplicity, I chose to use the pulse width output which outputs the range according to the following equation:

$$Range = \frac{PulseWidth}{147\mu s} inches$$

This formula was determined experimentally by setting up the circuit as seen in figure 1.

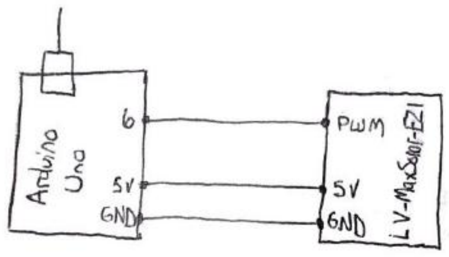

Figure 1: Circuit diagram for ultrasonic rangefinder interface with arduino

The rangefinder was secured and the resulting pulse width was analyzed at distances away from the sensor at 5, 10, 15, 20, and 25 inches. Through analysis roughly with a tape measure, it was determined that the sensor was in fact unable to measure any distance closer to it than 5.6 away. Furthermore, the accuracy of the PWM measuring technique was confirmed as each measurement was within a ±0.5 inches of where it was according to the tape measure, and that error was likely sourced from human error. Thus, it was determined that the provided transfer function was sufficient for the ultrasonic rangefinder, and as expected, the rangefinder measures all distances closer than 5.6 inches as such. For bench top testing there was little use in having the range of the rangefinder be much larger than 25 inches, thus it was decided to cap the sensor's range there. Furthermore, for use in controlling the motors, the output needed to be changed to be between 0 to 1023 which was done via a mapping function which mapped 5.6-25 inches to 1023-0, causing the DC motor in velocity control to have the highest rpm when an object is the closest to it.

1.1.2 Stepper Motor: Position Control

The Mercury Motor SM-42BYG011-25 Stepper Motor can easily be controlled with a DRV8825 Stepper Motor Driver. The circuit diagram for connecting the stepper motor to the stepper motor driver can be seen in figure 2. Interfacing with the Stepper Motor Driver can be done in one of two ways. The first is by setting the direction pin to high or low to determine the direction, and then oscillate setting the step pin on the driver from high to low at a specified timestep to move the motor. While this method of control works, it is far easier to utilize a downloadable library titled ”BasicStepperDriver” which allows for the stepper motor to be driven forward at a specified rpm or have a specified angle of rotation from where it currrently is. With the library implemented it was possible to use a power supply set at 12-15V to control the stepper motor to specified positions. A function was utilized which converted input signals from 0 to 1023 to specific angles from 0 to 360 by remembering previous positions and moving according to the current position and the previous position. The integrated code for controlling the stepper motor with the ultrasonic rangefinder can be seen in Appendix 6.1.1.

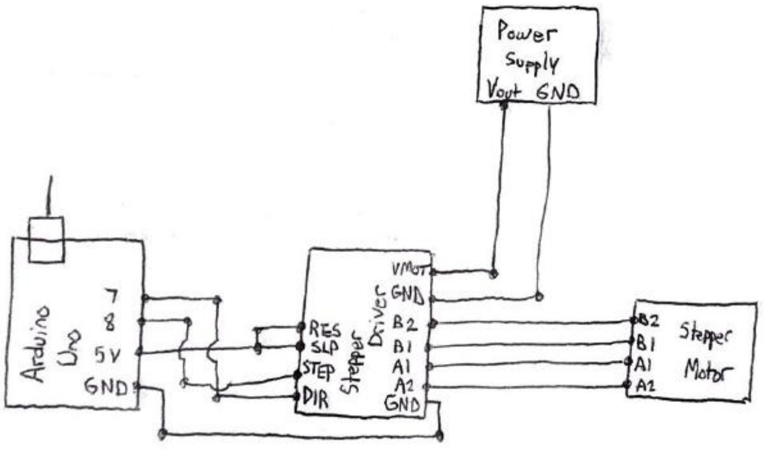

Figure 2: Circuit diagram for stepper driver and stepper motor interface with arduino

1.2 MRSD Project

Prior to the beginning of the semester I went through the process of learning ROS and understanding how it would be utilized for our project. I did this by going through all the beginner ROS tutorials and reading through some of the presentations on the project course canvas site. With this understanding, I was able to implement the docker environment set up by Kaushik on my system and verify that the ROS environment functions with the provided packages that are accessible on our GitHub. Once our software was set up, Sundaram, Anthony, and I went about getting trained in the machine shop by Tim in order to machine an interface to get our Atrascys camera mounted on the VESA mount we purchased. An image of the camera mounted can be seen in figure 3.

I then set about researching, designing, and prototyping an adapter (looking into ways of securing the mount, types of screws to use with the iliac crest in the hip, methods of reducing degrees of freedom, etc.) that could be screwed into our sawbone pelvis and hold our marker array, however our sponsor were able to provide us with one of their old Navio Single-Screw mounts which we can use instead to screw into the pelvis which can be seen in figure 4. We still need to develop and machine an adapter for the end-effector of the robot manipulator to interface with the acetabular reamer handle. This adapter needs to interface rigidly with the robot and reamer as well as hold a marker array for the Atrascys camera to track it’s position and with help from the absolute encoders in the robot arm, determine the location of the base link of the robot arm with regard to the pelvis location. The preliminary design of this adapter system can be seen in figure 5, however it is not

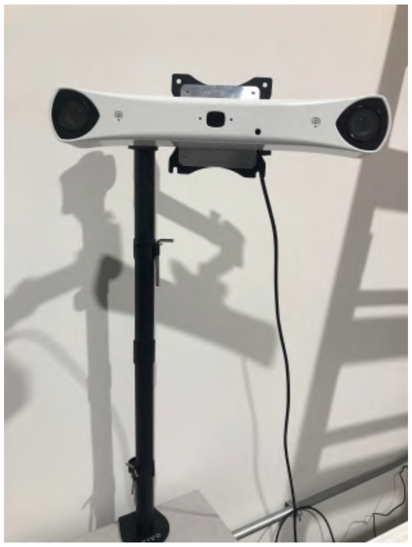

Figure 3: Camera mounted to VESA mount with custom interface

completed as we do not currently have mounting information for the reamer handle. I helped to set up our lab space in B512. As a group with the other team in B512, we organized the tables in the room to maximize space for the Franka robots in the autonomy course, and aided with setting up the Vention tables for the autonomy course in order to guarantee a space and table for our robot arm. We also had the opportunity to pick up our robot arm and get trained on it, however we realized during the training that it was not compatible with ROS currently and thus the arm may not be usable for our application. This issue will be discussed in further detail in the challenges section of this report.

2 Challenges

2.1 Sensors and Motors Lab

One big challenge with regard to the sensors and motors lab for me personally was trying to determine why my stepper motor was not being powered when properly hooked up to the power supply. I spent a large chunk of time using a digital multimeter testing whether specific pins had voltage across them and determining if current was flowing, which it was not to the motor. It turned out that I needed to solder the connectors to the stepper motor driver pins, which when done powered the stepper motor. Another issue is that the stepper motor was jittery and inaccurate when updated at a high frequency. This was due to the stepper motor being sent a new position when it was not done moving to a previously set position, leading to it moving a certain number of steps from where it currently was rather than where it was supposed to be. This was largely fixed through the utilization of a delay between when new commands can be sent to the stepper motor, allowing it to reach previously set positions before it is sent new position commands.

Some team-wide issues came with the integration of our full system. The serial communication between the GUI and the arduino code needed to be standardized and initialized at the same states. Furthermore, the way in which our finalized circuit was initially wired was a rats nest and need to be rewired with longer jumper wires that I brought from home. The biggest issue we spent debugging

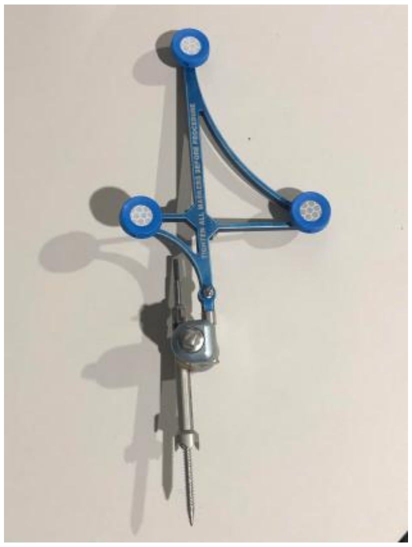

Figure 4: Navio single-screw mount

as a team was with regard to the light sensors which when implemented on a separate piece of code perfectly detected the lux as expected, but had a jittery measurement when implemented in our final arduino code. This turned out to be a function of the servo motor drawing too much current and thus the current output from the light sensor was inaccurate. We fixed this issue by connecting the servo motor to the constant 5V power supply.

The majority of the challenges from this lab came not from the individual development of ways to work with sensors and motors, but from the integration of all our work together. Potentially these issues could have been better mitigated by discussing and outlining our plan for the code prior to working individually on different aspects of the system. The finalized circuit used in the demonstration can be seen in figure 6.

2.2 MRSD Project

One challenge with the project for me was not having specific dimensions and information necessary for the accurate design and prototyping of the marker mounts. While we eventually got information on the mounting holes for the end-effector, we were unable to get accurate dimensions for the the acetabular reamer assembly and thus could not fully design an adapter for the end-effector and begin getting it manufactured. Figure 5 demonstrates this as there is a plate missing from the front which would be designed once further information became available. The most major challenge we faced this semester was everything with regard to the robot arm. The robot manipulator we were receiving this quarter came a week later than we anticipated, and once it arrived and we were trained in the use of the arm, we learned that the arm was incompatible with ROS and would not be compatible until this summer when an API is made available. This has led us to needing to talk with our sponsor and deciding that they will look into getting us a similar arm that we can use for our project instead of the manipulator we received. Furthermore, we are planning on reaching out to other professors in the Robotics Institute to see if there are any robot manipulators that would be willing to lend to us for our project.

Some other smaller issues that we faced this quarter is with regard to not having a Vention

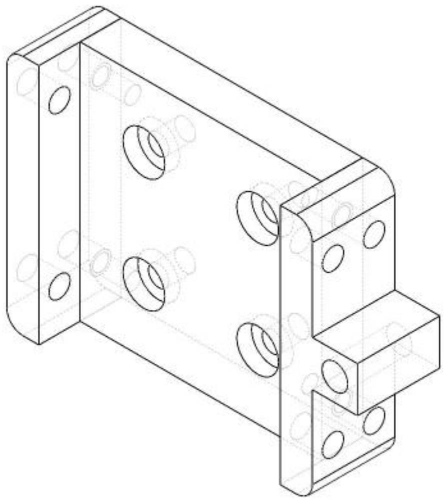

Figure 5: Prototype end-effector and reamer handle interface

table to set up the robot arm once it arrived as well as not having space for the robot arm in the lab initially. Professor Kroemer should be helping us with securing a Vention table, and we reorganized our lab space to allow for all the Franka arms in the Robot Autonomy course as well as our and another teams robot arms. We also dealt with our MRSD computer not working for a brief period due to an SSD being disconnected which was not allowing us to boot into Ubuntu. Finally, I personally had issues with enabling my Nvidia driver to work on my personal computer's dual boot Ubuntu which led to several days of working to fix the resulting issues.

3 Team Work

3.1 Sensor and Motors Lab

We all worked on separate parts of the sensor and motors lab initially until it came time to integrate all the parts together, at which point we worked simultaneously to find and fix issues prior to the presentation. Anthony worked on the force sensitive resistor, the servo motor, and set up the initial state machine and serial communication code for the arduino. Gunjan worked on setting up the Qt GUI and collaborated with Anthony to standardize the serial communication between the GUI and the arduino. Kaushik worked on the potentiometer, the DC motor, and helped figure out how to control the DC motor with PID position control. Sundaram worked on the ambient light sensor, the DC motor, and helped figure out how to control the DC motor with PID velocity control.

3.2 MRSD Project

For the first part of this project everyone put in a lot of good work, even despite the dearth of time to work on the project in general. Anthony worked primarily on updating the website, designing and machining the camera mount, and researching control methodologies that we could use with the robot manipulator. Gunjan focused primarily on everything related to the Atrascys camera, including getting it functioning as well as understanding how the data and transfer functions

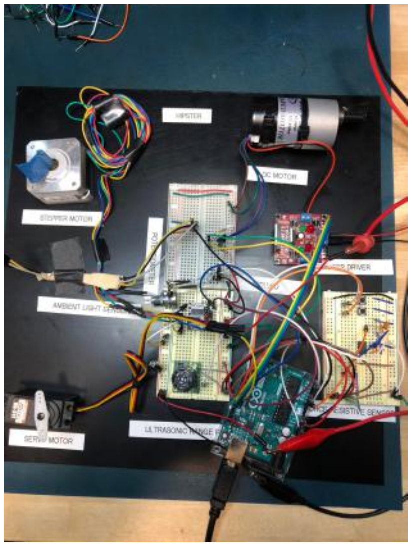

Figure 6: Final circuit

could be converted to be useful in ROS. Kaushik helped to set up the ROS docker environment and generated documentation for setting it up and pulling all custom packages we make from github. He also helped Gunjan with setting up the Atrascys camera and set up a simulation environment in Gazebo. Sundaram helped to keep us all on track with regards to the schedule as well as interfaced with our sponsor as our primary contact for the robot arm. He also focused on configuring the MoveIt ROS package for the URDF we received of the robot manipulator.

4 Plans

4.1 Sensor and Motors Lab

Unfortunately, the lessons we have learned from our utilization of the ultrasonic rangefinder, light sensor, potentiometer, and force sensitive resistor are not very applicable to our MRSD project. The biggest thing we gained from working on this lab is familiarity with the process of integrating software and hardware and some of the issues that come with that. Furthermore, our robot manipulators are likely to use some sort of DC motors with absolute encoders however, so the practice in learning how to implement PID control on a DC motor will be helpful down the line. Furthermore, while we may not be using a force sensitive resistor, there is a chance we use a load cell to determine the force experienced by the end-effector of our robot manipulator, thus having experience with force determining circuits may end up being useful.

4.2 MRSD Project

Our plan for the future hinges largely on when we receive a new robot manipulator for the project. Should Smith Nephew be able to procure us another manipulator to use, hopefully it will be here soon and we can begin configuring the ROS environment to work with that manipulator prior to it's arrival. In terms of designing the end-effector mount, as soon as I have more information with regards to this new manipulator and as soon as we get access to the acetabular reamer assembly, I can finalize and 3D print prototypes of the mount and verify it's efficacy before moving to the machine shop and creating a final mount. Furthermore, I plan to help more on the ROS side of the project more in the future as I have an interest in learning more about the simulation and control of the robot arm, especially considering that the project will be heavy in the utilization of ROS up until the point that we receive our new manipulator.

5 Quiz

5.1 Reading a datasheet

- The sensor's range is ±3.6 g in the typical case and a range of ±3.0 g in the minimal case.
- The sensors dynamic range is 7.2 g in the typical case and a dynamic range of 6.0 g in the minimal case.
- The capacitor C_{DC} is 0.1 µF and it decouples the accelerometer from noise on the power supply. It accomplishes this by smoothing changes in the voltage as high frequencies and transient currents flow through the capacitor to circuit ground instead of through the accelerometer. Essentially, the capacitor provides voltage to the accelerometer should the power supply dip, and absorbs energy (voltage) should the power supply voltage increase.
- The equation for the sensor's transfer function in the typical case where V is the output voltage and a is the input acceleration (measured in g's) is as follows: $V = (0.3V/g) * a + 1.5V$
- Based on the typical dynamic range of 7.2 g and an FSO% of 0.03%, the largest expected nonlinearity error in g would be ±0.0216 g.
- The data sheet has a conversion from Noise Density to rms Noise as such:

  $$rmsNoise = NoiseDensity * (\sqrt{BW * 1.6})$$

  Which can be solved given our bandwidth of 25 Hz and a noise density of 150 µg / Hz rms:

  $$rmsNoise = 150\frac{\mu g}{\sqrt{Hz}} * \sqrt{25Hz * 1.6} = 948.7\mu g = 9.5 * 10{-}4g$$

- A relatively easy experimental way of determining the RMS noise experimentally would be through using an oscilloscope, capturing data at a specific point, and then analyzing that data in MATLAB. In order to do this we would need to assume that the testing room is a perfect 25 °C and the input voltage to the accelerometer is a constant 3 V. Furthermore, we would assume that when the accelerometer is not being touched, it maintains a consistent mean of 1.5 V. With these assumptions, the accelerometer could be hooked up to a 3V power supply, and the output could be hooked up to an oscilloscope. The accelerometer should be placed flat on a table and undisturbed. Data should be collected via the oscilloscope and converted into a .csv file which could be then imported into MATLAB. In MATLAB, the data should be analyzed and the function y = rms(x) would provide the experimental root mean squared error of the accelerometer at whatever frequency we were testing at.

5.2 Signal conditioning

- Filtering
  - There are many potential issues with the utilization of a moving average filter as a software filter. One potential issue is that it introduces a lag to your signal, which would be exemplified by a step input to the filter being characterized as a linear rise instead. Another issue is that it does not reject outliers and instead includes them in the average over a long period as the window is recalculated.
  - Similar to the moving average filter, the median filter also has many potential issues. As a result of the the way in which a median filter is calculated through moving windows and finding medians, it is quite expensive and complex to compute which could lead to memory issues if the filter is used for a long period. Furthermore, while it does reject one time outliers, a median filter on it's own would not reject outliers that persist for a long time, and would necessitate experimentation to determine a window size to reject outliers that persist.
- Op-Amps
  - This circuit does have a solution and it can be found when V_1 is the reference voltage and V_2 is the input voltage. This leads to the following system of equations:
    $$0 = -1.5(1+\frac{Rf}{Ri}) - V_{\text{ref}}(\frac{Rf}{Ri})$$
    $$5 = 1(1+\frac{Rf}{Ri}) - V_{\text{ref}}(\frac{Rf}{Ri})$$
    Which when solved leads to a solution of $\frac{Rf}{Ri} = 1$ and $V_{\text{ref}} = -3V$.
  - This circuit does not have a solution and this can be demonstrated through an analysis of the system of equations when V_1 is the reference voltage and V_2 is the input voltage, though it also does not give a solution when those are inverted. Since -2.5 V to 2.5 V is a dynamic range of 5 V, and the output dynamic range is also 5V, it would seem as is only an offset would be necessary. But this necessitates a gain of 0, which would lead to the reference voltage being discounted, leading to the calibration being impossible.

5.3 Control

- A digital input for the proportional term could be determine with an encoder as well as the desired position. By subtracting these two terms from one another you could get the error with which to multiply by the proportional gain. A digital input for the integral term could be found by taking the error for the proportional input, multiplying by the sampling timestep, and adding it to a constant summation term in order to generate an integral of the error over time which could then be multiplied by the integral gain. Finally, a digital input for the derivative term could be found by taking the current encoder position subtracted by the previous encoder position and divided by the timestep to generate a derivative-esque term which could be multiplied by the derivative gain.
- If the control is sluggish we would want to increase the proportional gain to decrease the rise time.

- If the system has significant steady-state error we would want to increase the integral gain as an integral term helps with eliminating steady-state error.
- If the system has overshoot we would want to increase the derivative gain as an increased derivative gain would compensate for the increased velocity of the motor's movement due to a higher proportional gain leading to less or eliminated overshoot.

6 Appendix

6.1 Arduino Code

6.1.1 Code for Ultrasonic Rangefinder and Stepper Motor

```
#include <BasicStepperDriver.h>

/** Pin Assignments */
const int sensorPushButton = 4;
const int motorPushButton = 5;
const int servoPin = 9;
const int FSR_Pin = A0;
const int USRF_Pin = 6;
const int Stepper_Dir_Pin = 7;
const int Stepper_Step_Pin = 8;

int motorInput;

// Ultrasonic Range Finder Variables
float distance;
float new_distance;
float duration;
const float usrf_scale = 147;
const int usrfWindowSize = 5;
double usrf_window[usrfWindowSize];
float usrf_sma = 0;

// Stepper Motor Variables
const int Stepper_Steps = 200;
const int Stepper_RPM = 240;
const int Stepper_Microsteps = 1;
BasicStepperDriver stepper(Stepper_Steps,Stepper_Dir_Pin,Stepper_Step_Pin);
int previousStepperInput = 0;
int stepperInputDiff;
int stepperRotation;
const int stepperDegrees = 720;
const float stepperUpdate = 500;
float stepperTimer;
```

```
void setup() {
  Serial.begin(9600);
  pinMode(USRF_Pin,INPUT);
  stepper.begin(Stepper_RPM,Stepper_Microsteps);
  stepperTimer = micros();
}

void loop() {
   // Get distance and convert to motor input
   distance = getDistance();
   motorInput = d2input(distance);

   // Update stepper if it has been enough time
   if((millis()-stepperTimer)>stepperUpdate){
    updateStepper(motorInput,previousStepperInput);
    stepperTimer = millis();
   }
}

float getDistance(){
  duration = pulseIn(USRF_Pin,HIGH); //Finding duration of PWM Pulse
  new_distance = duration/usrf_scale; //Converting from duration of PWM Pulse to distanc
  if (distance > 100){
    distance = 100;
  }

  for (int i = 0; i <usrfWindowSize;i++){ //Implementing a 5 point SMA
    if (i == usrfWindowSize-1){
      usrf_window[i] = new_distance;
    }
    else{
      usrf_window[i] = usrf_window[i+1];
    }
  }
  usrf_sma = usrf_sma+(1/(double)usrfWindowSize)*(usrf_window[4]-usrf_sma);
  return usrf_sma;
}
int d2input(double d){ //Conversion factor from distance to 0->1024
  double input = map(d,5.6,100,1024,0);
  if (input > 1024){
    input = 1024;
  }
  else if (input < 0){
    input = 0;
  }
  return input;
}

void updateStepper(int input, int previous_input){ // Update stepper
  stepperInputDiff = input - previous_input;
  stepperRotation = stepperInputDiff*((double)stepperDegrees/1024);
  Serial.println(stepperRotation);
  stepper.rotate(stepperRotation);
  previousStepperInput = input;
}
```

6.1.2 Completed Arduino Code

```
/** Libraries */
#include <BasicStepperDriver.h> // From Library "StepperDriver" by Laurentiu Badea
#include <PinChangeInterrupt.h> // From Library "PinChangeInterrupt" by NicoHood
#include <PinChangeInterruptBoards.h>
#include <PinChangeInterruptPins.h>
#include <PinChangeInterruptSettings.h>
#include <Servo.h>
#include <string.h>
#include <util/atomic.h> // For the ATOMIC_BLOCK macro


/** Macros */
#define LIGHTSENSORPIN A2 //Ambient light sensor reading
#define light_window 50

/** Pin Assignments */
const int sensorPushButton = 4;
const int motorPushButton = 5;
const int servoPin = 10;
const int FSR_Pin = A0;
const int USRF_Pin = A5;
const int Stepper_Dir_Pin = 7;
const int Stepper_Step_Pin = 8;
//Motor encoderPins
const int ENCA = 2;
const int ENCB = 3;
//Motor signal pins
const int PWM = 9;
const int IN1 = 11;
const int IN2 = 12;
//Potentiometer pins
```

```
const int pot = A1;

/** Global Variables */
// State Machine Variables
enum MotorState {SERVO, DC_MOTOR_POS, DC_MOTOR_VEL, STEPPER};
enum SensorState {FSR, ULTRASONIC, POT, LIGHT, GUI};
static MotorState motorState;
static SensorState sensorState;
static SensorState prevSensor;
const unsigned long debouncingPeriod = 250; // ms
unsigned long motorButtonTimer;
unsigned long sensorButtonTimer;
int motorInput;

// GUI Variables
int guiInput;
const char delim[2] = ",";

// FSR Variables
const int fsrSampleFrequency = 1000; // sampling frequency in hertz
const int fsrWindowLength = 25; // length of window (samples)
double window[fsrWindowLength]; // window of moving-average filter
const double Vin = 3.3; // voltage input to voltage divider circuit
const double saturationVoltage = 2.7; // volts
const double maxAnalogVoltage = 5; // volts
double force;
unsigned long fsrTimer;
double prevFSRValue = 0;

// Ambient Light Sensor Variables
int a = 0;
int a_analog = 0;
float sum = 0;
float sum_analog = 0;
float readings[light_window];
float readings_analog[light_window];
float avg = 0;
float avg_analog = 0;
float lux;
int lux_analog;
long lightTimer = 0;

int m = 0;
float sum_motor = 0;
float readings_motor[light_window];
float avg_motor = 0;

// Ultrasonic Range Finder Variables
float distance;
float new_distance;
float duration;
const float usrf_scale = 147;
const int usrfWindowSize = 10;
double usrf_window[usrfWindowSize];
float usrf_sma = 0;

// Potentiometer Variables
int raw = 0;
int target_map = 0;
float angle = 0;
float potAngle = 0;
float potRaw = 0;

// Servo Variables
Servo myservo;
const int servoUpdateFrequency = 10; // update every 100 ms
unsigned long servoTimer;

// Stepper Motor Variables
const int Stepper_Steps = 200;
const int Stepper_RPM = 240;
const int Stepper_Microsteps = 1;
BasicStepperDriver stepper(Stepper_Steps, Stepper_Dir_Pin, Stepper_Step_Pin);
int previousStepperInput = 0;
int stepperInputDiff;
int stepperRotation;
const int stepperDegrees = 360;
const float stepperUpdate = 100;
float stepperTimer;

// DC Motor Variables - Velocity Control
// DC Motor Velocity Control Variables
int posPrev = 0;
float previousError = 0;
char incomingByte;
volatile int encoderValue = 0;
float edot = 0;
float proportional = 0.5; //2; //k_p = 0.5
float integral =0.55; //3 //k_i = 3
float derivative = 0.0001;//0.12; //k_d = 1
float controlSignal = 0; //u - Also called as process variable (PV)
float errorIntegral = 0;
float deltaT;
//int pos;
long previousTime;
long currentTime;

// DC Motor Variables - Position Control
long prevT = 0;
float eprev = 0;
float eintegral = 0;
float deltaTpos;
float kp = 25;
float kd = 1.2;
float ki = 0.5;
float currT = 0;
float target = 0;

// Setup********************************************************************************
void setup()
{
  // Attach pins
  pinMode(FSR_Pin, INPUT);
  pinMode(sensorPushButton, INPUT_PULLUP);
  pinMode(motorPushButton, INPUT_PULLUP);
  attachPCINT(digitalPinToPCINT(sensorPushButton), changeSensor, FALLING);
  attachPCINT(digitalPinToPCINT(motorPushButton), changeMotor, FALLING);
  pinMode(USRF_Pin, INPUT);
  pinMode(LIGHTSENSORPIN,  INPUT);


  // State machine setup
  sensorButtonTimer = 0;
  motorButtonTimer = 0;
  motorState = SERVO;
  sensorState = POT;


  // GUI setup
  prevSensor = FSR;
  Serial.begin(115200);
  guiInput = 0;

  // Sensor setup
  fsrTimer = micros();
  force = 0;

  // Motor setup
  motorInput = 0;
  setupServo(myservo, servoPin);
  stepperTimer = micros();
  stepper.begin(Stepper_RPM, Stepper_Microsteps);
  pinMode(ENCA,INPUT);
  pinMode(ENCB,INPUT);
  pinMode(PWM,OUTPUT);
  pinMode(IN1,OUTPUT);
  pinMode(IN2,OUTPUT);
  attachInterrupt(digitalPinToInterrupt(ENCA),encoder,RISING);
  currentTime = micros();
  previousTime = micros();
}


// loop*****************************************************************************
void loop()
{
  // Read from GUI
  // Reads 3 comma-delimited numbers. First number determines whether GUI is enabled,
  // second is which motor is enabled (same order as enum declaration above), third numb
  if (Serial.available()) {
    String incomingMessage = Serial.readString();
    char buffer[incomingMessage.length() + 1];
    incomingMessage.toCharArray(buffer, incomingMessage.length() + 1);
    char *token;
    token = strtok(buffer, delim);
    int inputs[3] = { -1, -1, -1};
    for (int i = 0; i < 3; i++) {
      if (token != NULL) {
        inputs[i] = atoi(token);
        token = strtok(NULL, delim);
      } else {
        //Serial.println("Invalid Message");
        i = 3;
      }
    }
    if (inputs[0] != -1 && inputs[1] != -1 && inputs[2] != -1) {
      for (int i = 0; i < 3; i++) {
        switch (i) {
          case 0:
            if (inputs[i] == 0) {
```

```
//              Serial.println("Gui Disabled");
              disableGUIControl();
            } else {
//              Serial.println("Gui Enabled");
              enableGUIControl();
            }
            break;
          case 1:
//            Serial.println(inputs[i]);
            guiChangeMotor((MotorState) inputs[i]);
            break;
          case 2:
            if (sensorState == GUI) {
//              Serial.println(inputs[i]);
//              controlSignal = 0;
//              errorIntegral = 0;
              guiInput = inputs[i];
            }
            break;
        }
      }
    }
  }

  // Read FSR Sensors
  if ((micros() - fsrTimer) > ((unsigned long)1000000) / fsrSampleFrequency) {
    force = getForce();
    fsrTimer = micros();
  }
  // Read USRF Sensor
  distance = getDistance();

  //Read ambient light sensor - Lux output for GUI
  if ((micros() - lightTimer) > 100000) {
    lux = ambientLightRead();
    lux_analog = ambientLightAnalog();
    lightTimer = micros();
  }

  //Read Potentiometer angle
  potRaw = getPot();
  angle = getAngle();


  // Determine which sensor is the input to the motor
  switch (sensorState)
  {
    case FSR:
      motorInput = f2input(force);
      break;
    case ULTRASONIC:
      motorInput = d2input(distance);
      break;
    case POT:
      motorInput = potRaw;
      break;
    case LIGHT:
      motorInput = lux_analog;
      break;
    case GUI:
      motorInput = guiInput;
      break;
  }
  // Drive chosen motor
  switch (motorState)
  {
    case SERVO:
//      Serial.println("SERVO");
      if ((micros() - servoTimer) > ((unsigned long)1000000) / servoUpdateFrequency) {
        updateServo(myservo, motorInput);
        servoTimer = micros();
      }
      break;
    case STEPPER:
//      Serial.println("STEPPER");
      if ((millis() - stepperTimer) > stepperUpdate) {
        updateStepper(motorInput, previousStepperInput);
        stepperTimer = millis();
      }
      break;
    case DC_MOTOR_POS:
//      Serial.println("DC_MOTOR_POS");

        currT = micros();
        deltaTpos = ((float) (currT-prevT)/1000000);
        if (sensorState != GUI) {
          int target_map = map(motorInput, 0, 1023, 0, 300);
//          motorInput = 0.25 * target_map;
//          Serial.println(motorInput);
      }
  if (deltaTpos > 0.045) {


//    Serial.print("Target: ");
//    Serial.println(motorInput);

    // time difference
  long currT = micros();
//  float deltaTpos = ((float) (currT - prevT))/( 1.0e6 );
  prevT = currT;

  // Read the position in an atomic block to avoid a potential
  // misread if the interrupt coincides with this code running
  // see: https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifie
  int pos = 0;
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
    pos = encoderValue;
  }
//  Serial.print("Current Position: ");
//  Serial.println(pos);
  // error
  target = motorInput/4;
  int e = pos - target;

  // derivative
  float dedt = (e-eprev)/(deltaTpos);

  // integral
  eintegral = eintegral + e*deltaTpos;

  // control signal
  float u = kp*e + kd*dedt + ki*eintegral;
//  Serial.print("u: ");
//  Serial.println(u);

  // motor power
  float pwr = fabs(u);
  if( pwr > 255 ){
    pwr = 255;
  }

  // motor direction
  int dir = 1;
  if(u<0){
    dir = -1;
  }

  // signal the motor
  setMotor(dir,pwr,PWM,IN2,IN1);


  // store previous error
  eprev = e;

//  Serial.print(target);
//  Serial.print(" ");
//  Serial.print(pos);
//  Serial.println();
//  Serial.print(u);
//  Serial.print(" ");

}
      break;


    case DC_MOTOR_VEL:
      if (sensorState != GUI)
      {
        motorInput = map(motorInput, 0, 1023, 0, 100);
//        Serial.println(motorInput);
      }

      int pos_vel = encoderValue;

      long currentTime = micros();
      deltaT = ((float) (currentTime-previousTime)/1000000);
//      Serial.print(deltaT);

      if (deltaT > 0.01)
      {
        float velocity = (pos_vel - posPrev)/deltaT;
        posPrev = pos_vel;
        previousTime = currentTime;

        float v1 = velocity*60/90;
//        Serial.println(v1);
        float error = motorInput - v1;
        float u = calculatePIDVel(error);

        int dir = 1;
        if (u<0)
        {
          dir = -1;
        }
        int pwr = (int) fabs(u);
        if(pwr > 255)
        {
          pwr = 255;
        }
        else if(pwr < 0)
        {
          pwr = 0;
        }
        setMotor(dir,pwr,PWM,IN1,IN2);

        sum_motor = sum_motor - readings_motor[m];
        readings_motor[m] = v1;
        sum_motor =  sum_motor+v1;
        m = (m+1)%light_window;
        avg_motor = sum_motor / light_window;
//        Serial.println(avg_motor);
      }



      break;


  }

  // Sends outgoing messages to GUI to inform gui of sensor readings and motor positions
  Serial.print("u,");Serial.println(distance); // u is for ultrasonic sensor
  Serial.print("f,");Serial.println(force); // f is for force sensor
  Serial.print("a,");Serial.println(lux); // a is for ambient light sensor
  Serial.print("p,");Serial.println(angle); // p is for potentiometer reading
  // m is for motor, and the numbers after that indicate which motor is switched on (0-3
  // only sends motor reading for the motor currently selected, otherwise the reading is
  // Example: m,0,512,0,0,0, which means that the servo motor is selected and has an inp
  switch (motorState)
  {
    case SERVO:
      Serial.print("m,");Serial.print(0);Serial.print(",");Serial.print(motorInput);Seri
      break;
    case DC_MOTOR_POS:
      Serial.print("m,");Serial.print(1);Serial.print(",");Serial.print(0);Serial.print(
      break;
    case DC_MOTOR_VEL:
      Serial.print("m,");Serial.print(2);Serial.print(",");Serial.print(0);Serial.print(
      break;
    case STEPPER:
      Serial.print("m,");Serial.print(3);Serial.print(",");Serial.print(0);Serial.print(
      break;
  }
}

// State Machine Functions**************************************************************
void changeSensor()
{
  if ((millis() - sensorButtonTimer) > debouncingPeriod) {
    switch (sensorState)
    {
      case FSR:
        sensorState = ULTRASONIC;
//        Serial.println("ULTRASONIC State");
        break;
      case ULTRASONIC:
        sensorState = POT;
//        Serial.println("POT State");
        break;
      case POT:
        sensorState = LIGHT;
//        Serial.println("LIGHT State");
        break;
      case LIGHT:
        sensorState = FSR;
//        Serial.println("FSR State");
        break;
      case GUI:
        break;
    }
    sensorButtonTimer = millis();
  }
}

void enableGUIControl()
{
  if (sensorState != GUI) {
    motorInput = 0;
    prevSensor = sensorState;
    sensorState = GUI;
  }
}

void disableGUIControl()
{
  sensorState = prevSensor;
}

void changeMotor()
{
  if ((millis() - motorButtonTimer) > debouncingPeriod) {
    switch (motorState)
    {
      case SERVO:
        motorInput = 0;
        motorState = DC_MOTOR_VEL;
//        Serial.println("DC_MOTOR_POS State");
        break;
      case DC_MOTOR_POS:
        motorInput = 0;
        motorState = DC_MOTOR_VEL;
        errorIntegral = 0;
//        Serial.println("DC_MOTOR_VEL State");
        break;
      case DC_MOTOR_VEL:
        motorInput = 0;
        motorState = STEPPER;
//        Serial.println("STEPPER State");
        break;
      case STEPPER:
        motorInput = 0;
        motorState = SERVO;
//        Serial.println("SERVO State");
        break;
    }
    motorButtonTimer = millis();
  }
}

void guiChangeMotor(MotorState state)
{
  if (state >= 0 && state <= 3) {
    motorState = state;
    if (state == STEPPER || state == SERVO) {
      setMotor(0, 0, PWM, IN1, IN2);
    }
  }


}

// Sensor Functions*********************************************************************
// FSR Functions------------------------------------------------------------------------
/**
   Updates the Force Sensitive Resistor reading and gives the estimated force applied on
   @return the Force (N) that is applied to the FSR
*/
double getForce()
{
  return v2F(updateFilter(min(saturationVoltage, (analogRead(A0) / 1023.0) * maxAnalogVo
}

/**
   Maps a given force to a 10-bit integer to be used as motor input
   @param f the force read by the FSR
   @return the motor input (10-bit integer)
*/
int f2input (double f)
{
  return (f / v2F(saturationVoltage)) * 1023;
}

/**
   Updates the moving-average filter/window by recalculating the filters output with the
   and then updates window with the given value.
   @param v the most recent sampled voltage
   @return the most recent filter output
*/
double updateFilter(double v)
{
  prevFSRValue = ((prevFSRValue * ((double)fsrWindowLength)) - window[0] + v) / ((double
  for (int i = 0; i < fsrWindowLength - 1; i++) {
    window[i] = window[i + 1];
  }
  window[fsrWindowLength - 1] = v;
  return prevFSRValue;
}

/**
   Converts voltage (volts) to force (newtons). Uses the 2nd-order calibration curve tha
   part 1 of the project.
   @param v the voltage (volts)
   @return the force (newtons)
*/
double v2F (double v)
{
  if (v < 0.956 && v > -0.001) { // 0.956 volts is when the calibration curve is actuall
    return v * 0.59 / 0.956; // linear increase
  }
  return 1.923 * (v * v) + -3.677 * v + 2.348;
}

// USRF Functions-----------------------------------------------------------------------
float getDistance() {
//  duration = pulseIn(USRF_Pin, HIGH); //Finding duration of PWM Pulse
  new_distance = 5*analogRead(USRF_Pin) / 9.8;
//  Serial.println(5*analogRead(USRF_Pin) / 9.8);

//  duration = 10; // Used for testing to get rid of delay from PulseIn fx
//  new_distance = duration / usrf_scale; //Converting from duration of PWM Pulse to dis
  if (distance > 25){
    distance = 25;
  }
  for (int i = 0; i < usrfWindowSize; i++) { //Implementing a 5 point SMA to eliminate n
    if (i == usrfWindowSize - 1) {
      usrf_window[i] = new_distance;
    }
    else {
      usrf_window[i] = usrf_window[i + 1];
    }
  }
  usrf_sma = usrf_sma + (1 / (double)usrfWindowSize) * (usrf_window[4] - usrf_sma);
  return usrf_sma;
}
int d2input(double d) { //Conversion factor from distance to 0->1024
  double input = map(d,5.6,25,1024,0);
  if (input > 1023) {
    input = 1023;
  }
  else if (input < 0) {
    input = 0;
  }
  return input;
}
```

```
//Ambient Light sensor function
float ambientLightRead(){
      sum = sum - readings[a];
      float reading = analogRead(LIGHTSENSORPIN);
      readings[a] = reading;
      sum =  sum+reading;
      a = (a+1)%light_window;
      avg = sum / light_window;

//      Serial.print("Reading: "); Serial.print(reading);

//      Serial.print("Avg. Reading: "); Serial.print(avg);

      float volt = (avg*5.0)/1023.0;
//      Serial.print(" Volt:"); Serial.print(volt);

      float curr = pow(10,6)*(volt/100000);
//      Serial.print(" Curr:"); Serial.print(curr);

      float lux = pow(10, 0.1*curr);
//      Serial.print(" Lux: "); Serial.println(lux);
      return lux;
  }

//Ambient sensor analog output
float ambientLightAnalog(){
    sum_analog = sum_analog - readings_analog[a_analog];
    float reading_analog = analogRead(LIGHTSENSORPIN);
    readings_analog[a_analog] = reading_analog;
    sum_analog =  sum_analog+reading_analog;
    a_analog = (a_analog+1)%light_window;
    avg_analog = sum_analog / light_window;

//    Serial.print("Avg. Reading: "); Serial.print(avg_analog);
    return avg_analog;
  }

//Potentiometer Functions----------------------------------------------------------------

float getAngle(){

  int raw = analogRead(pot);
  int potAngle = map(raw,0,1023,0,300);
  int target_map = map(raw, 0, 1023, 0, 600);
  float target = 0.25 * target_map;
  return potAngle;
}

float getmotorAngle(){
  int raw = analogRead(pot);
  int target_map = map(raw, 0, 1023, 0, 600);
  float target = 0.25 * target_map;
  return target;
}

float getPot(){
  int raw = analogRead(pot);
  return raw;
}

// Motor Functions*****************************************************************
// Servo Functions-----------------------------------------------------------------
/**
   Runs the functions needed to setup the servo
   @param s the servo object
   @param controlPin the pin controlling the PWM sent to the servo
*/
void setupServo(Servo s, int controlPin)
{
  s.attach(controlPin);
  s.write(0);
  servoTimer = micros();
}

/**
   Updates the servo position based on the input
   @param s the servo object
   @param input the 10-bit number representing the input to the servo (maps input to bet
*/
void updateServo(Servo s, int input)
{
  s.write((int)map(input, 0, 1023, 0, 180));
}

// Stepper Functions---------------------------------------------------------------
void updateStepper(int input, int previous_input) {
  stepperInputDiff = input - previous_input;
  stepperRotation = stepperInputDiff * ((double)stepperDegrees / 1023);
  stepper.rotate(stepperRotation);
  previousStepperInput = input;
```

```
}


//DC Motor Velocity Control

//Velcoity Control main PID Loop (calculatePID(errorValue) gets called insider here)

void velocityControl(){

  int pos = encoderValue;
  float velocity = (pos - posPrev)/deltaT;
  posPrev = pos;
  previousTime = currentTime;

  float v1 = velocity*60/90;
//  Serial.print("ActualVelocity: ");
//  Serial.println(v1);
//  Serial.print("MotorInput: ");
//  Serial.println(motorInput);
  float error = motorInput - v1;
  float u = calculatePIDVel(error);
//    Serial.print("u: ");
//  Serial.println(u);
  int dir = 1;
  if (u<0){
    dir = -1;
  }
  int pwr = (int) fabs(u);
  if(pwr > 255){
    pwr = 255;
  }
  else if(pwr<0){
    pwr = 0;
  }
//  Serial.println("Inside velocity");
  setMotor(dir,pwr,PWM,IN1,IN2);

}

//Velcity control PID Function

float calculatePIDVel(float errorValue)
{
  edot = (errorValue - previousError) / deltaT; //edot = de/dt - derivative term
  errorIntegral = errorIntegral + (errorValue*deltaT);
  Serial.println(errorIntegral);
  if (errorIntegral > 480) {
    errorIntegral = 480;
  } else if (errorIntegral < -480) {
    errorIntegral = -480;
  }
  controlSignal = (proportional * errorValue) + (derivative * edot) + (integral * errorI
  if (controlSignal > 260) {
    controlSignal = 260;
  } else if (controlSignal < -260) {
    controlSignal = -260;
  }
  previousError = errorValue; //save the error for the next iteration to get the differe
  Serial.println(controlSignal);
  return controlSignal;
}

//Velocity and Position Control Encoder

void encoder() {
  if (digitalRead(ENCB) == HIGH) // if ENCODER_B is high increase the count
    encoderValue++; // increment the count

  else // else decrease the count
    encoderValue--;  // decrement the count

}

//General Motor Function

void setMotor(int dir, int pwmVal, int pwm, int in1, int in2)
{
//  Serial.print(pwmVal);
  analogWrite(pwm,pwmVal); // Motor speed
  if(dir == 1){
    // Turn one wayb
    digitalWrite(in1,HIGH);
    digitalWrite(in2,LOW);
  }
  else if(dir == -1){
    // Turn the other way
    digitalWrite(in1,LOW);
    digitalWrite(in2,HIGH);
  }
  else{
    // Or dont turn
    digitalWrite(in1,LOW);
    digitalWrite(in2,LOW);
  }
}
```