Individual Lab Report - 1

Autonomous Reaming for Total Hip Replacement

HIPSTER | ARTHuR

Kaushik Balasundar

Team C:

Kaushik Balasundar | Parker Hill | Anthony Kyu

Sundaram Seivur | Gunjan Sethi

February 10th 2022

Carnegie Mellon University

The Robotics Institute

Contents

1 Individual Progress **1**
1.1 Sensor and Motors Lab . . . 1
1.1.1 Potentiometer . . . 1
1.1.2 DC Motor : Position Control . . . 1
1.1.3 Other Contributions . . . 2
1.2 MRSD Project . . . 3

2 Individual Progress **3**

3 Challenges **4**
3.1 Sensors and Motors Lab . . . 4
3.2 MRSD Project . . . 4

4 Team Work **5**

5 Plans **6**
5.1 Sensor and Motors Lab . . . 6
5.2 MRSD Project . . . 6

6 Sensors and Motors Lab Quiz **7**

7 Code **11**

1 Individual Progress

1.1 Sensor and Motors Lab

My tasks and responsibilities for the Sensors and Motors Lab were in interfacing the potentiometer and also performing position control of the DC motor. In addition, I spent time assisting Sundaram integrate and tune the PID controller for the velocity controller since some segments of our code were coupled. Moreover, I also assisted in the overall integration and debugging of the circuit software with the GUI. Outlined below are the steps I took to do the aforementioned tasks.

1.1.1 Potentiometer

Interfacing the potentiometer included reading the analog pin connected to the middle pin of the potentiometer. The remaining pins on the extreme left and right were connected to the power (5V) and ground (GND) pins of the Arduino. Once the readings were acquired from the analog pins, the value between 0 and 1023 was mapped between 0 and 300 through a linear interpolation using Arduino's map function. These limits were chosen since they were the respective minimum and maximum angles that the potentiometer could rotate through, as mentioned on the datasheet.

1.1.2 DC Motor : Position Control

My second contribution was in performing position control of the DC Motor. The encoder attached to the DC Motor needed to be connected to two interrupt pins (2 and 3 on the Arduino) to read the encoder counts with time. There are 90 ticks for a complete revolution of the output shaft as given by the datasheet. Using this, I determined that four ticks produce a degree of rotation. The resolution can be improved on using both channels in the motor's encoder. Each time the subroutine was called, I determined the direction of rotation by measuring the state of the other pin - HIGH indicates rotation in one direction (increment encoder tick reading) and LOW indicates rotation in the other direction (decrement encoder tick reading), based on how the motor is connected to the driver. Once I determined the target angle and the corresponding encoder ticks needed, a PID control loop was used to output a control signal that achieved the desired position using the desired number of ticks. I started tuning the controller by first initializing the proportional, derivative and integral gains to 0, and only increasing the proportional gain 'P' until I received reached a desirable rise time. The I increased the derivative term 'D' to reduce oscillations and finally the increased the integral term 'I' values to reduce the steady state error. I also incorporated an atomic block in the code to force the compiler to finish assigning the encoder ticks to local variables before moving onto the next interrupt subroutine, which sometimes caused invalid data to be assigned.

The next part involved me integrating the potentiometer with the the DC Motor for position control. For better visibility, each degree of rotation of the potentiometer causes 2 degrees of rotation of the main shaft of the sensor. The potentiometer angle value was converted to the required number of ticks by dividing the number of degrees by four. So, a rotation of 360 degrees would translate to having the encoder increment by 90 ticks. The number of ticks was the target that the PID control loop tried to converge to. In our integrated code, each motor expects an input between 0-1024 and their respective functions performed the necessary conversions to rotate by the required

number of degrees in position or by the desired amount in velocity as in the case of the DC motor's velocity control.

1.1.3 Other Contributions

Apart from this, I assisted Sundaram with the DC motor's velocity control since some segments of code were common. In addition to this, I also assisted in the overall integration and testing of the code with the rest of the team.

The circuit diagram for the potentiometer integrated with the DC Motor and driver is shown in Figure 1. The complete integrated circuit is as shown in Figure 3. The graphical user interface of our system is as shown in Figure 2.

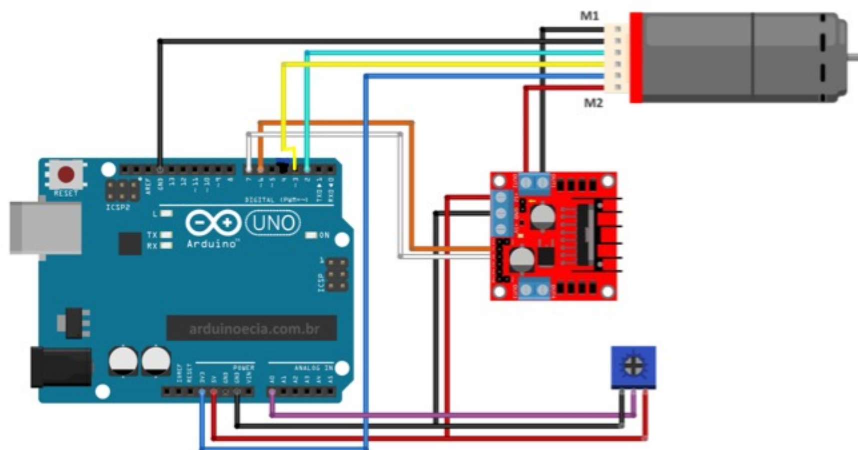

Figure 1: Circuit diagram for Potentiometer and DC Motor

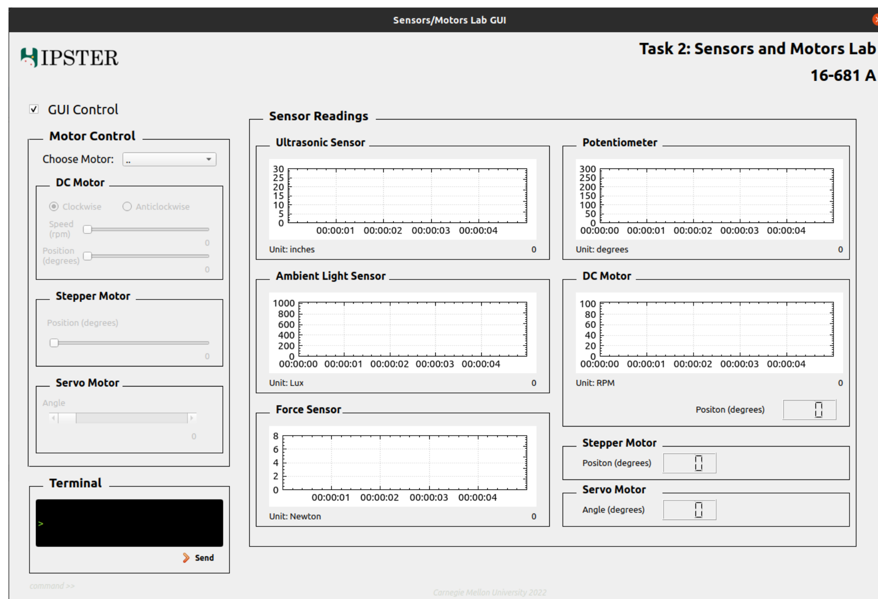

Figure 2: Graphical User Interface

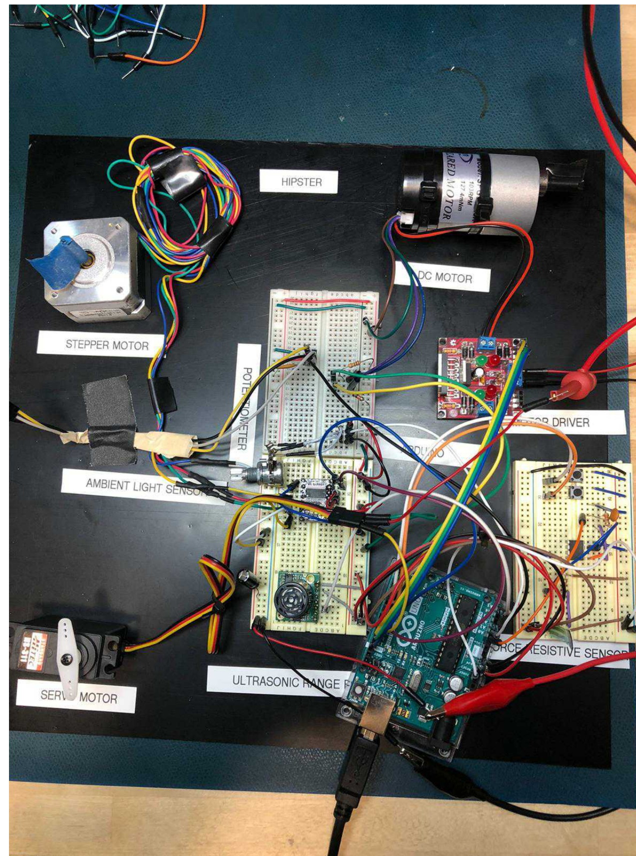

Figure 3: Integrated Hardware Circuit

1.2 MRSD Project

2 Individual Progress

As a part of the MRSD project, I worked on the following aspects:

1. **Setting up of a development environment:** Our team decided to use Docker to establish a synchronous and uniform development environment that we could all work on. As a part of this, I was responsible for creating the docker registry, setting up the ROS environment, creating documentation for general setup and also ensuring that I educate the team on the best practices to get maximum benefit of the platform. This environment is still work in progress, and I am responsible for ensuring that the issues encountered are addressed in a timely manner.

2. **Setting up of a GitHub workflow:** It was evident from previous MRSD projects as well as from our own on working on the sensors and motors lab that planning for effective subsystem integration is key to the success of the project. To make our workflow modular and to facilitate effective integration, I setup a set of repositories that correspond to the various subsystems in our project that can be easily developed individually and effectively integrated. The various repositories that our team would be working abide by standard ROS practices and are as follows:

- arthur_description: Houses the URDF and the RViz visualization files.
- arthur_gazebo: Houses the Gazebo simulation models, worlds and launch file.
- arthur_control: A ROS package for the controls functionality of the arthur robot.
- arthur_bringup: An integration package intended to bring up all the major functionality.
- arthur_moveit: A ROS Package for the MoveIt! integration of the robot arm.

3. **Setting up Gazebo Simulation**: We obtained a bare-bone URDF of the Kinova arm we were provided from the manufacturer. However, many modifications had to be made to the URDF to make it possible to visualize and model on Gazebo. Following this, the joint trajectory controllers had to be setup and finally a preliminary MoveIt! package was configured for the arm. I collaborated with Sundaram on this part. The joint trajectory controllers are compatible for execution with Moveit!.

3 Challenges

3.1 Sensors and Motors Lab

Some challenges that we faced are as follows:

1. **PID Tuning**: Initially, after writing the control loop for position control, tuning the PID parameters was a challenge since the motor shaft position was varying almost sporadically for small changes in the proportional gain. The motor shaft as a result was oscillating the PID and it was difficult to dampen the oscillations even with increasing the derivative term. We then realized that the sampling rate of our PID controller was much higher than that of the encoder which caused these issues. By adding an atomic block to prevent encoder value update issues, and also by reducing the sampling frequency, we were able to get more reliable encoder readings and tuning it after this point was relatively straightforward. However, debugging this issue was a challenge.

2. **System Integration**: While each team member was able to finish their individual code fairly quickly, it was a challenge to integrate these segments of code reliably. There were some conflicting pin assignments which had to be reassigned. We also realized that there was some integral wind-up that had to be dealt with when we changed motors. Debugging this was quite time consuming. We also realized that the servo motor was drawing too much current which was interfering with the analog inputs that some of our sensors were receiving. We solved this by supply power to the Servo motor externally through the power supply. Since we had substantial number of Serial print commands to communicate the sensor data with the GUI, the code in general took longer to execute and our PID parameters previously tuned had to be modified with this current setup. However, our team put in a cohesive effort to methodically debug these issues to get it working on time for the demonstration.

3.2 MRSD Project

1. **Docker Setup**: I was setting up a Docker container for ROS for the first time so there were a lot of issues such as enabling hardware acceleration with the host Graphics card, ensuring

reliable GUI permissions and interfacing and setting up the necessary packages. We are still having issues making some controls packages working in the Docker environment which I am still working towards. Ensuring proper documentation so that each team member is able to setup this environment on their systems was also time-consuming, but worth the effort.

2. **Robot Arm URDF**: We were provided with a developmental version of the URDF by Kinova, which had a lot of issues that needed to be debugged. Moreover, it was not compatible with Gazebo for simulation. I had to include the necessary tags in the URDF to ensure this compatibility. In addition, I also had to setup the controllers from scratch for the joints.

3. **Robot Arm API Support:** The robot arm we received from Kinova unfortunately has no API support until June 2022. We will unfortunately switch to another arm until this support is made available. We are currently asking around RI for an alternative arm to use. Possible arms to use would be the Franka Panda, Kinova Gen 3 and UR5. This would also mean that some of the work done earlier with the URDF of the current arm would be redundant. Nevertheless, it was a good learning experience.

4 Team Work

Our team members divided our work in the following manner:

1. **Kaushik**: I worked on position control of the DC Motor and Potentiometer. Also contributed to aiding in the software system integration and debugging. For the MRSD project I worked on setting up the Docker environment, basic framework for our code, a Gazebo simulation environment, and aided in the setting up of controller. Currently, I am working on the perception sub-component.

2. **Sundaram**: He worked on velocity control of the DC Motor and on the ambient light sensor. He also contributed to aiding in the hardware and software system integration. For the project, he worked on the fabrication of the camera mount, setting up the controllers for the arm in simulation and configuring the Moveit! package in simulation.

3. **Anthony**: He worked on the Servo motor and Force Resistive Sensor that included the creation of a transfer function plot and analog filtering. He played a major role in setting up a framework for code integration with a state machine and communication with the GUI. Finally, he aided in system integration. For the project, he worked on the MRSD website, camera mount design and fabrication and some research on the selection of the control system to be used on the arm.

4. **Parker**: He worked on the Stepper motor and Ultrasonic Range Finder (USRF). He also aided in hardware and software system integration of the circuit. For the project, he worked on the camera mount fabrication, researching and designing the marker mounts, picking up the robot arm, and learning ROS.

5. **Gunjan**: She worked primarily on the Graphical User Interface (GUI) and also aided in the software system integration. For the project, she worked primarily with the Atrycys camera integration with ROS while ensuring proper linking of C++ libraries. This allowed her to

gain a deeper understanding of CMake. In addition, her contributions were also towards maintaining documentation and Jira for project management.

5 Plans

5.1 Sensor and Motors Lab

We will not be pursuing any of the components in the sensors and motors lab since it is not of relevance to the project. This is due to the fact that we already have a robot arm hardware and sensor setup over which we will be developing our software.

5.2 MRSD Project

In the upcoming weeks, I plan to work on the following:

1. Open3D Point Cloud Registration: I will be exploring local registration using techniques such as Iterative Closest Point (ICP) to integrate a sparse surface model of the acetabulum with the pre-operatively obtained dense model of the pelvis.

2. Fixing bugs in the Docker environment: While a large portion of the setup is done, there are still some minor bugs that need to be fixed for each team-member to have a smooth experience working with the environment.

3. Possible changes to simulation environment due to the arm being incompatible: The robot arm we received from Kinova unfortunately has limited API support. As a result, we will need to change our arm until this support is made available. We are currently asking around the Robotics Institute for an alternative arm to use. Possible arms to use would be the Franka Panda, Kinova Gen 3 and UR5. We will need to modify the simulation environment accordingly.

6 Sensors and Motors Lab Quiz

1. Question 1: Reading a Datasheet

    - What is the sensor's range?

      The sensor's range is [-3:3g]

    - What is the sensor's dynamic range?

      The dynamic range is 6g

    - What is the purpose of the capacitor CDC on the LHS of the functional block diagram on p. 1? How does it achieve this?

      The capacitor ensures that the power supply remains more stable. It acts as a filter to get rid of high frequency noise that might interfere with the sensor's readings. If the voltage falls below the desired voltage, the capacitor will behave like a small battery and discharge to provide the necessary electrical power to keep the voltage constant.

    - Write an equation for the sensor's transfer function.

      $V_{out} = 1.5V + 300(\frac{mV}{g}) * a$

    - What is the largest expected non-linearity error in g?

      0.3% of full-scale reading = 0.003 $\times$ ± 7.2g = $\pm$ 0.0216g

    - What is the sensor's bandwidth for the X- and Y-axes?

      1600 Hz

    - How much noise do you expect in the X- and Y-axis sensor signals when your measurement bandwidth is 25 Hz?

      Noise$_{rms}$ = 150 * $\sqrt{25 \times 1.6}$
      Noise$_{rms}$ = 948.68μ g

    - If you didn't have the data-sheet, how would you determine the RMS noise experimentally? State any assumptions and list the steps you would take.

      I would first mount the sensor to a flat surface or an equivalent motion-cancelling flat surface. In this configuration, any readings it would detect would be noise. Then, I would record multiple measurements of the sensors and compute its average. Finally,

taking the square root of that mean will give me the RMS noise.

2. Question 2 : Signal Conditioning

   - Name at least two problems you might have in using a moving average filter.

     (1) The moving average filter is just a poor low pass filter. This means that the input value is smoothed out to remove sharp changes, making the filter sluggish to sudden changes in the input. (2) It also has slow roll-off and poor stop-band attenuation characteristics. (3) Moreover, when filtering high frequency noise, the moving average filter needs a larger window making it lag and rendering it unsuitable for low-latency applications.

   - Name at least two problems you might have in using a median filter.

     (1) If the window size used is small and there are multiple outliers that represent the extremeties, it will increase the error. (2) The process of finding the median required sorting which makes the entire process more computationally expensive. In the bast-case scenario, the time complexity is $nlogn$ where n refers to the window width. It is evident that as n increase the computation cost will also increase significantly.

   - In the following questions, you want to calibrate a linear sensor using the circuit in Fig. 1 so that its output range is 0 to 5V. Identify in each case: 1) which of V_1 and V_2 will be the input voltage and which the reference voltage; 2) the values of the ratio $\frac{R_f}{R_i}$ and the reference voltage. If the calibration can't be done with this circuit, explain why.

     Case 1:

$$\frac{V_1 - V_2}{R_i} = \frac{V_2 - V_{out}}{R_f} \implies V_{out} = (V_2 - V_1)\frac{R_f}{R_i} + V_2 \tag{1}$$

Putting $V_1 = V_{in}$ and $V_2 = V_{out}$ in the above equation:

$$\frac{R_f}{R_i} = -2k\Omega \tag{2}$$

This is not possible. Reversing the input and reference voltages and substituting in (1), we get:

$$\frac{R_f}{R_i} = 1k\Omega \tag{3}$$

Thus, V_2 is the input and V_1 is the reference.

Case 2:

Putting $V_1 = V_{in}$ and $V_2 = V_{out}$ in the above equation:

$$\frac{R_f}{R_i} = -1k\Omega \tag{4}$$

Putting $V_1 = V_{in}$ and $V_2 = V_{out}$ in the above equation:

$$\frac{R_f}{R_i} = 0k\Omega \tag{5}$$

In the second case, calibration cannot be done with the given gain and offset.

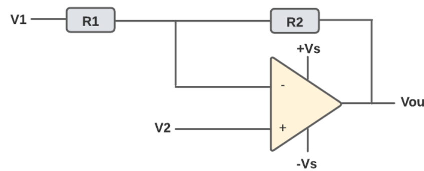

Figure 4: Opamp Circuit

3. Question 3: Controls

- If you want to control a DC motor to go to a desired position, describe how to form a digital input for each of the PID (Proportional, Integral, Derivative) terms.

  For the proportional term, the input is the difference between the target position and current position, called the error. For the integral term, the input is a running sum of errors multiplied by the time-step of the control loop. Finally, for the derivative term, the input is the rate of change of error given by the difference between previous position current position divided by the time-step of control loop.

- If the system you want to control is sluggish, which PID term(s) will you use and why?

  To increase the rise time of the system, I would increase the proportional term until the system responds fast enough to converge at the desired value. Increasing the proportional term too much however could result in overshoots, which need to be dampened out using the derivative term.

- After applying the control in the previous question, if the system still has significant steady-state error, which PID term(s) will you use and why?

I would increase the integral term “I” since this acts like a memory that accumulates errors over time and produces an integral term output that eliminates the steady state error.

- After applying the control in the previous question, if the system still has overshoot, which PID term(s) will you apply and why?

  As mentioned above, I would increase the derivative term “D” which acts as a virtual damper and slows down the gain as it approaches the desired value which in turn reduces the overshoot.

7 Code

```
// ** Code for position control with potentiometer ** //

// ** Snippet of integrated 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


/** Pin Assignments */
//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;

// Potentiometer Variables
int raw = 0;
int target_map = 0;
float angle = 0;
```

```
float potAngle = 0;
float potRaw = 0;


// 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()
{


  // 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()
{
 // Other team member's code

  //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;

      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;


    break;
  }

  switch (motorState)
  {
    case SERVO:
      Serial.print("m,"); Serial.print(0);
      Serial.print(","); Serial.print(motorInput);
      Serial.print(","); Serial.print(0); Serial.print(",");
      Serial.print(0); Serial.print(","); Serial.println(0);
```

```
      break;
    case DC_MOTOR_POS:
      Serial.print("m,"); Serial.print(1); S
      erial.print(","); Serial.print(0); Serial.print(",");
      Serial.print(motorInput); Serial.print(","); Serial.print(0);
      Serial.print(","); Serial.println(0);
      break;
    case DC_MOTOR_VEL:
      Serial.print("m,"); Serial.print(2);
      Serial.print(","); Serial.print(0);
      Serial.print(","); Serial.print(0);
      Serial.print(","); Serial.print(avg_motor);
      Serial.print(","); Serial.println(0);
      break;
    case STEPPER:
      Serial.print("m,"); Serial.print(3);
      Serial.print(","); Serial.print(0);
      Serial.print(","); Serial.print(0);
      Serial.print(","); Serial.print(0);
      Serial.print(","); Serial.println(motorInput);
      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();
  }
}


//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;
}


//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 * errorIntegral); //final sum, proportional term also calculated here
  if (controlSignal > 260) {
    controlSignal = 260;
  } else if (controlSignal < -260) {
    controlSignal = -260;
  }
  //save the error for the next iteration to get the difference (for edot)
  previousError = errorValue;
  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);
  }
```

```
}
```