Sensor and Motor Control Lab

Individual Lab Report 1

By Shivang Baveja (andrewID: sbaveja)

October 13, 2017

Team C : FlySense

Shivang Baveja

Harikrishnan Suresh

Nicholas Crispie

Joao Fonseca Reis

Sai Nihar Tadichetty

Overview

For the Sensor and motor control lab, the team first identified the sensors. This was done based on experience and ease of integration. Following sensors were selected:

1. Ultrasonic Range Finder - LV-MaxSonar-EZ1: digital
2. Sharp IR distance sensor: analog
3. Potentiometer: analog
4. Force-sensitive sensor: digital

As part of the objectives, we had to control following motors:

1. DC motor: Both position and speed control
2. Stepper motor: position control
3. Servo motor: position or speed (we chose position)

The whole system was implemented as a state machine, with only one of the motor control active at any point. The state was cycled using the pushbutton. The system starts in State0 in which all the motors are off. 5 states were defined which are explained via the following table:

State	Sensor	Motor
0	Not active	Not active
1	Ultrasonic sensor (distance)	DC motor position
2	Potentiometer(voltage)	DC motor speed
3	Sharp IR (distance)	Servo position
4	Force Sensor	Stepper position

A simple GUI was designed which allowed us to see the sensor and motor values for each of the state separately. It also allowed to override the sensor, by giving direct input to any of the motor control directly again depending upon the current state.

Contribution (Source code in appendix)

I was responsible for:

1. Setting up the state machine based on pushbutton.
2. Ultrasonic sensor, potentiometer
3. DC motor: position and speed control

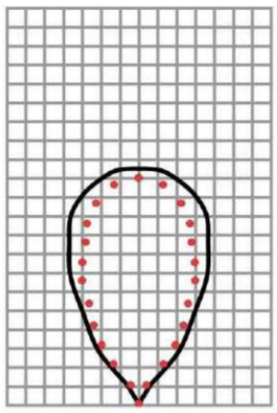
State Machine:

The state machine interface code was reused from the previous microcontroller lab familiarization. It was designed such that other team member's code should be easily integrated later with it.

Ultrasonic Sensor interface:

The Max Sonar sensor can be interfaced via analog line, serial data or PWM input. PWM input was selected to be easiest and less prone to noise. An initial test with the sensor was done to ascertain how well the sensor works and if there are any issues which needs to be resolved. There were following observations:

1. Overall the sensor readings were found to be accurate up-to 3cm.
2. Overall sensor was found to be reliable up-to a range of 2m.
3. The sensor readings weren't reliable below 20cm which was also mentioned in the datasheet
4. The sensor is cone of operation, in which the sensor sees the obstacles. The beam angle is around +-15 deg. This can be understood slightly better from the following image:



5. There were spikes in the received values of 12 cm every 1 or 2 seconds.

The above issues were accounted in the code by using a median filter to filter out spikes. Also any value below 20cm was discarded as the reliable operational range was above 20cm.

Potentiometer:

The potentiometer was easy to setup and gave reliable values. The output voltage of 0 to 5V from potentiometer was read using 10bit ADC which gave a resolution of 0.004V.

DC Motor:

For DC motor position and speed control, a basic PID control was developed. It was important to have a fixed loop rate to accomplish this. A loop rate of 50Hz was found to be sufficient to control the position and speed.

For the position control, the encoder tics were converted to angle. The error was calculated using by subtracting the current angle from the target. The error was integrated at every loop run and dt was estimated correctly for this purpose. The angular rate was also calculated using the consecutive angle values. This was required for the d-component which was mainly used as a damper.

For the integrator component some special considerations were taken. The integrator was not allowed to run when the control input has already saturated. Also, the integrator component was limited to half the control input range.

Control Input = P_gain * angle_error + I_gain * integrator - D_gain * Angular rate

For Speed control, only PI controller was used. This was done as a reliable estimate of angular acceleration was difficult to estimate and would have been noisy. Like for the position control, error and integral of error were calculated.

Control Input = P_gain * angle_rate_error + I_gain * integrator

The gains were tuned by developing a basic interface to tune the values in real time. This greatly decreased the tuning time as opposed to hard coding the gains. I was assisted by Joao in tuning the PID gains for position and speed control.

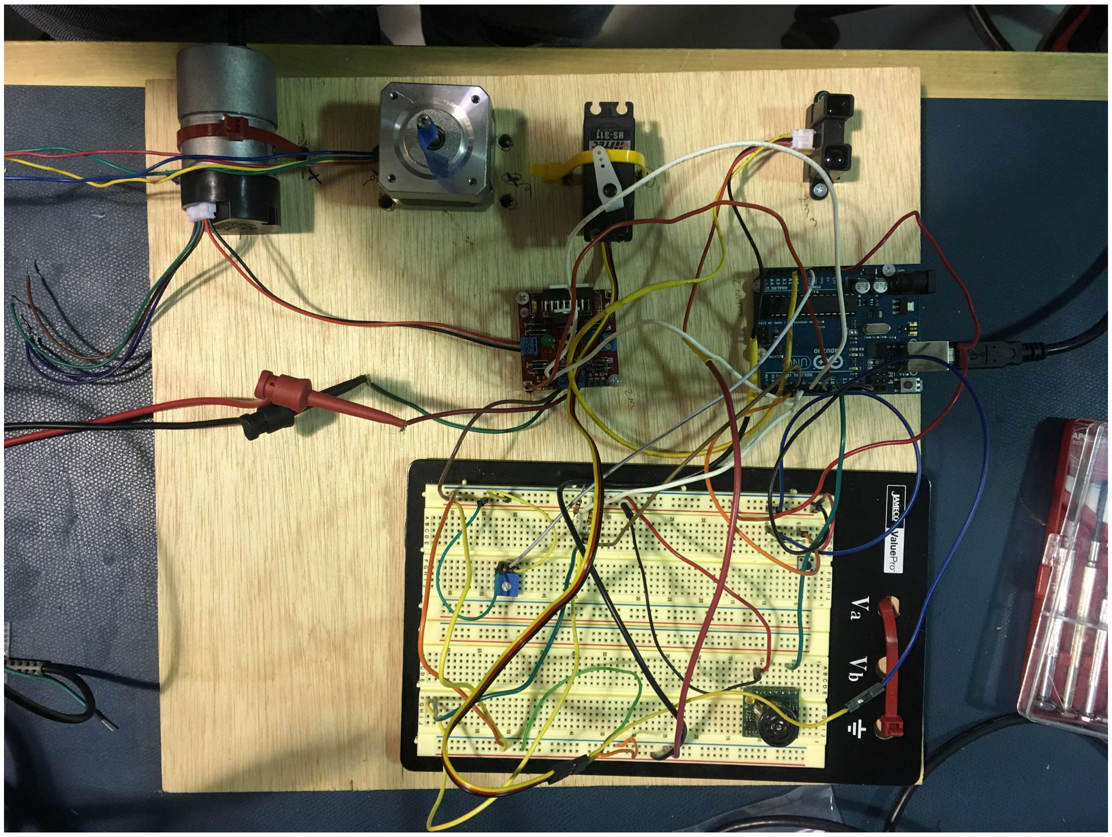
Results:

Position control:
steady state error of +-7 deg (which was getting cleared in about 4 seconds)
Rise time: found to be about 1 second
Speed control:
steady state error of about +-5 deg/s (it was variable over the range)
Rise time: found to be about 1-1.5 seconds

Challenges faced:

1. DC motor position control: Initially I was trying to do it without the D component but that was causing high oscillations when P gain was increased too much. Also, it wasn’t possible to use the Integrator as it was causing sustained low-frequency oscillations. So, d-gain component was introduced which allowed increasing the P-gain further and helped damp the oscillations caused by integrator.
2. Integrating all the sensors and motor code posed certain issues as there were some conflicts between different sensor/motor combinations.
3. The whole circuit was re-wired to make it cleaner. During this the DC motor control pins were switched which revered the directions. Luckily, there was an old code snippet stored on github which we knew was working. This helped us narrow down the issue to a hardware issue.

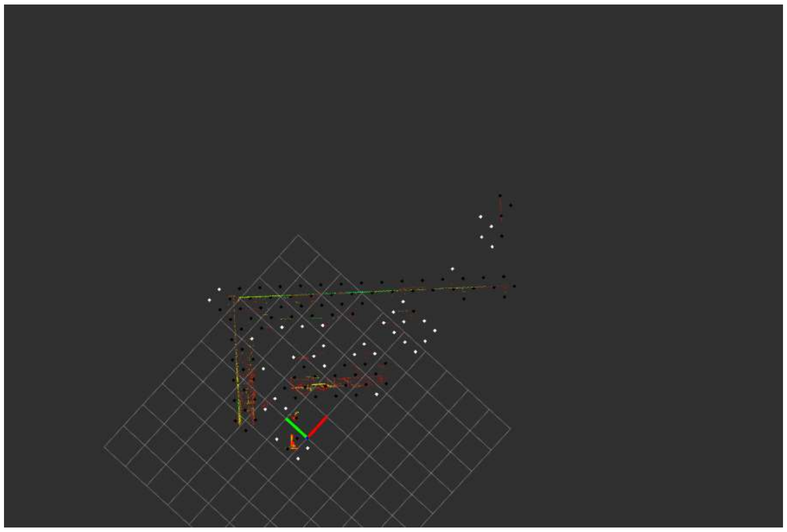
Hardware Setup:



Team Contributions:

1. Joao Fonseca Reis: was responsible for GUI, Hardware integration
2. Nick Crispie: Force sensor interface and stepper motor control. Wiring the system
3. Harikrishnan Suresh: was responsible for IR distance sensor interface and servo motor control
4. Nihar: GUI development and establishing serial protocol.

MRSD Project contributions:



My primary role in the project is in sensing and hardware, along with Hari. The most important component in the sensing system is the LIDAR, which will be used to detect and track obstacles in real time. For the fall validation experiment and the scale model testing (in quadcopter), we are planning to use Velodyne Puck VLP-16.

We were successful in interfacing the LIDAR with ROS and observing the raw point cloud data in Rviz. Setting up the sensor had some complications, which eventually turned out to be clashes in IP address. We have prepared a document on the various steps to be followed to get the sensor up and running in ROS, which could later be used by other teams working with the same sensor. The point cloud data produced by the LIDAR when placed inside the MRSD lab is shown in Figure 6.

We also tried out two ROS packages that take the 3D point cloud data and process it to find the obstacles in the environment. These were just trials on investigating whether the existing algorithms will be useful for our purpose or not.

velodyne_height_map - This ROS package uses a height map algorithm to take the point cloud data, detect the obstacles and clear spaces around. The results obtained after running this package are shown in Figure 7. The region with obstacles and clear spaces are replaced by a set of points in 3D, and can be distinguished. But it does not give a clear and direct segmentation of the environment. For pilots, such an obstacle data will prove to be ineffective unless we run another algorithm over this.

octomap - This ROS package implements a 3D occupancy grid mapping using the concept of octrees. To get this working, the launch file of octree was modified - frame of the map was made same as the frame of LIDAR, and the topic to which the map subscribed to was renamed to the topic published by the LIDAR. The resulting map is shown in Figure 8. The octomap produced a 2D occupancy grid map instead of 3D map, which calls for further modifications to the node.

Task 7 (Sensors and Motor Control Lab) Quiz

1. Reading a datasheet. Refer to the ADXL335 accelerometer datasheet (https://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf) to answer the below questions.
   - What is the sensor's range?

     →Typical Range: +-3.6g

     Min Range: +-3g

   - What is the sensor's dynamic range?

     →6g

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

     →Its function is to reduce the incoming ripple noise coming from the power supply and stabilize the voltage across it. Once the capacitor is charged it allows the DC component and resists the AC ripple in the supply voltage.

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

     → Vout = 1.5 V + (300 mV/g)*a

     where a is the input acceleration

   - What is the largest expected nonlinearity error in g?

     → Largest error due to non-linearity = 0.3% of full scale

     = +-0.018g

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

     →For x_{OUT}, y_{OUT} the typical noise level is 150 micro g / sqrt (Hz).

     Thus, for 25 Hz the expected noise level should be = 150 * 10^-6 * Sqrt(25)

     = 0.00015 * 5 = 0.000750g {when measuring in g's}.

   - How about at 0 Hz? If you can't get this from the datasheet, how would you determine it experimentally?

     → It's not available in the data sheet. We can do a static test to find the sensor's noise. To do this we can keep the sensor fixed to a surface which is static and we record readings of the sensors in each axis. Then we can calculate root mean square of that to get correct noise measure.

2. Signal conditioning
   - Filtering
     - What problem(s) might you have in applying a moving average?

→Moving average adds delay in the system which is hard to model when running any control algorithms. It reduces the spike in the system at the cost of losing certain information (like a spike which should have been recorded).

- What problem(s) might you have in applying a median filter?

→ The windows size of how many values to consider for the median filter must be varied according frequency of the outliers in the signal. It might let some spikes through if the window size is not right.
Also, the median filter takes a lot of memory as all the values must be stored and computation wise takes more processing compared to a moving average filter.

- Opamp
  - 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 which of V1 and V2 will be the input voltage and which the reference voltage, the value of the reference voltage, and the value of Rf/Ri in each case. If the calibration can't be done with this circuit, explain why.

    →
    Writing the basic opamp voltage equation,
    Vout = V2 + (Rf/Ri)*(V2-V1)

    Considering all the cases:
    - **If V2 is Vref, V1 is input,**
      - **V2 is positive, Vout is positive (not 0) when V1 is negative**
      - **V2 is negative, Vout is negative when V1 is positive**
    - **If V1 is Vref, V2 is input,**
      - **V1 is positive, Vout is negative (not 0) when V2 is negative**
      - **V1 is negative satisfies, both the negative and positive inputs. So, V1 must be negative.**

    - Your uncalibrated sensor has a range of -1.5 to 1.0V.

    → To achieve the output range (0 to 5v) mapped to input range (-1.5 to 1)
    V1 is Vref and V1=-3
    Rf/Ri=1

    - Your uncalibrated sensor has a range of -2.5 to 2.5V.

      Vout = V2 + (Rf/Ri)*(V2 + V1)

      5 = -2.5 + (Rf/Ri)*(-2.5 +V1)
      0 = 2.5 + (Rf/Ri)*(2.5 +V1)

Adding the two equations,
We get, (Ri/Rf)*V1=2.5

Since resistance can't be negative, V1 has to be positive. If V1 is positive we don't have a solution which maps to output (0 to 5V), as described above.

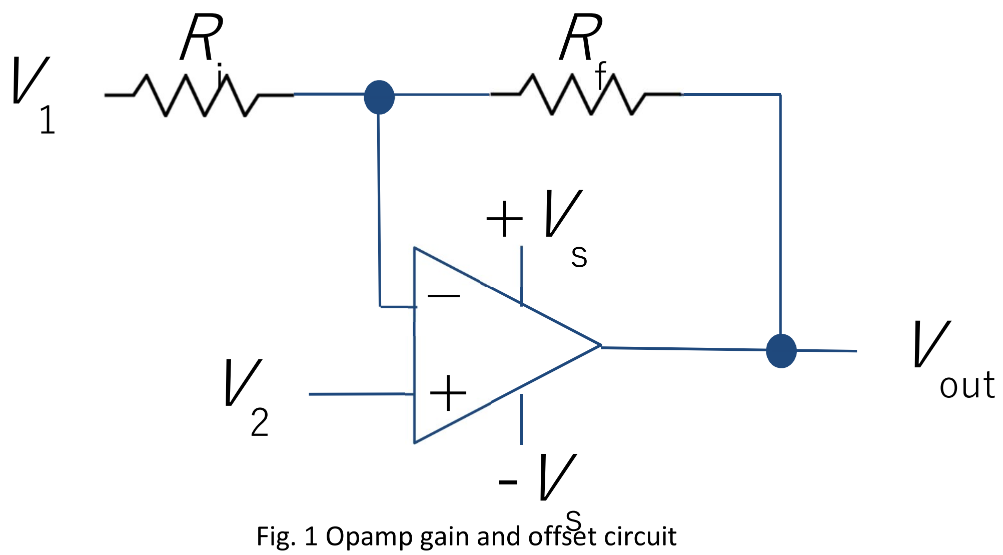


Fig. 1 Opamp gain and offset circuit

3. Control
   - 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.

     →Calculate error, error= Target_position – Current_position

     Integrate the error, but only till the control saturates
     Take differential of the error value.
     Control input = P_gain * error + I_gain* Integrator + D_gain * Error rate (differential of error)

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

   - 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?

→ **Increase the I-gain**

- After applying the control in the previous question, if the system still has overshoot, which PID term(s) will you apply and why?
  → **Decrease the d-gain or add some p-gain**

Appendix: Source Code

```
/*
 * Project course - Arduino Assignment
 * Microcontroller familiarization
 * Arduino UNO Pin # Component
 * A0 Potentiometer
 * 2 Button 0
 * 3 Button 1
 * 9 Red LED channel
 * 10 Green LED channel
 * 11 Blue LED channel
 *
 *
 *
 * Other websites referred:
 * https://www.arduino.cc/en/Reference/DigitalRead
 *
 *
 *
 */

//#include "PinChangeInt.h"
#include <Servo.h>
#include <medianFilter.h>
#include <Stepper.h>


//Nick's code
#define DIR 11
#define STEP 10


/****************************** Start of Macro definitions*************************************/
#define BAUD 38400
#define PIN_BUTTON0 (2)
#define ENCODER1    (3)
#define POTENTIOMETER (0)
#define ULTRASONIC_PIN  (8)
#define SERVO          (9)

//anticlockwise when looking at shaft from the front, when high given to F and low given to B
#define M_CTRL_F  (5)       //Motor control forward, goes to L1,
#define M_CTRL_B (7)        //Motor control backward, goes to L2
#define ENCODER2  (4)
#define SPEED_CONTROL_PIN   (6)

#define P_GAIN      (7.0)//(7.0)
#define I_GAIN      (1.0)//(0.0)
```

```
#define D_GAIN      (1.0)//(1.0)
#define INT_MAX     (100.0)//(150.0)

#define SPEED_I_GAIN    (0.7)
#define SPEED_P_GAIN    (2.0)
#define SPEED_INT_MAX   (150)

#define DEBOUNCE_DELAY  20     //micro sec

/******************************* End of MACRO definitions*******************************/

/******************************* Start of Type declarations************************************/
typedef enum
{
  STATE0=0,
  STATE1=1,
  STATE2=2,
  STATE3=3,
  STATE4=4
}STATE;

/********************************* End of Type Declrations*************************************/

/*******************************Start of Global variable section ********************/

/*
 * Hari's servo IR integration
 */
const int ir=A1;
//float IRinput[25];
float IRdistance;
float IRvolt,IRvolt1;
float tmppos;
int pos;
Servo myservo;
int servo_target=0;
float filtered_IR=0;
 /*************************/

/*
 * Nick's code
 */
 int fsrPin = A2;     // the FSR and 10K pulldown are connected to a0
int fsrReading;     // the analog reading from the FSR resistor divider
int fsrVoltage;     // the analog reading converted to voltage
unsigned long fsrResistance;  // The voltage converted to resistance, can be very big so make "long"
unsigned long fsrConductance;
long fsrForce;       // Finally, the resistance converted to force
```

```
bool stepper_at_pos = false;
float desired_angle = 0.0;
float desired_step = 0.0;
int step_size = 0;
int stepper_init=0;

float current_angle=0;

const int stepsPerRevolution = 800;  // change this to fit the number of steps per revolution
// for your motor

// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, DIR, STEP);
 /************************/

STATE state=STATE0;

int statusLedPin = 13; // LED connected to digital pin 13
int a=0;

int potReadPin=POTENTIOMETER;

int potValue=0;
float potVolt=0.0;
float potNorm=0.0;

long button1_press_start_time=0;
int button1_press_time=0;
int button1_rise_detected=0;
int val_button1 = 0;     // variable to store the read value

int button0_pin = PIN_BUTTON0;
long button0_press_start_time=0;
int button0_press_time=0;
int button0_rise_detected=0;
int val_button0 = 0;     // variable to store the read value

long loop_last_time=0;
int loop_run_flag=0;
int hz1_loop=0;
int hz5_loop=0;

// Serial data
#define MAX_BUFF_LEN 20
char data_buffer[MAX_BUFF_LEN];
char data[4];
int index=0;
int read_index=0;
int data_index=0;
int rx_count=0;
int data_ready=0;
int command_start=0;

String str,str2;
int start_parse=0;

int test_flag=0;

/*
 * Encoder specific variables
 */
volatile long encoder_count=0;
long last_encoder_count=0;

/*
 * Ultrasonic variables
 */
 const int ultraPin = ULTRASONIC_PIN;
 long ultraPulse, ultraInches;
long ultraCM;
medianFilter Filter,Filter_IR;
int filtered_ultra=0;
/*******************************End of golobal Variable section ****************************/

/*
 * PID control
 */
int control_input=0;      //0 sensor controlled, 1 user controlled

float error_pos=0;
int target_pos=0;
int current_pos=0;
float integrator=0;
long int pid_old_time=0;
float control_out=0;

float speed_target=0;
float speed_actual=0;
float speed_control_out=0;
float speed_integrator=0;
float speed_error=0;
float speed_I=SPEED_I_GAIN;
float speed_P=SPEED_P_GAIN;
int speed_dir=1;


float p_gain=P_GAIN;
float i_gain=I_GAIN;
float d_gain=D_GAIN;
float int_max=INT_MAX;
int d_component=0;
 /******************************/


/******************************Start of function section  *************************/


void change_state()
{
  control_input=0;
  switch(state)
  {
    case STATE0:
        state=STATE1;
        break;
    case STATE1:
        state=STATE2;
        break;
    case STATE2:
        state=STATE3;
        break;
    case STATE3:
        state=STATE4;
        break;
    case STATE4:
        state=STATE1;
        break;
  }

  Serial.print("State");
  Serial.println(state);
}

void button0_pressed()
{
    button0_press_start_time=millis();
    button0_rise_detected=1;
}

void encoder1_change()
{
    bool encoder1_val=PIND & B00001000;
    bool encoder2_val=PIND & B00010000;
    if(encoder1_val==encoder2_val)
    {
      encoder_count--;
    }
    else
    {
      encoder_count++;
    }
}

void stop_motor()
{
  analogWrite(SPEED_CONTROL_PIN,0);
}

void motor_forward_pulse(int pulse)
{
  if(pulse<0)
  {
    pulse = 0;
  }
  else if(pulse>255)
  {
    pulse = 255;
  }
  analogWrite(SPEED_CONTROL_PIN,pulse);
  digitalWrite(M_CTRL_F, HIGH);
  digitalWrite(M_CTRL_B, LOW);
}

void motor_backward_pulse(int pulse)
{
  if(pulse<0)
  {
    pulse = 0;
  }
  else if(pulse>255)
  {
    pulse = 255;
  }
  analogWrite(SPEED_CONTROL_PIN,pulse);
  digitalWrite(M_CTRL_F, LOW);
  digitalWrite(M_CTRL_B, HIGH);
}

int i=0;
int dir=1;

void run_motor_sweep()
{

  if(i>10)
  {
    i=0;
    if(dir==1)
    {
      dir=-1;
      target_pos=0;
      motor_forward_pulse(250);
      Serial.print(i);
      Serial.println("Forward");
    }
    else
    {
      dir=1;
      target_pos=100;
      Serial.print(i);
      motor_backward_pulse(250);
      Serial.println("Backward");
    }
  }
  else
  {
    i++;
  }
}

void run_motor_speed()
{
   if(speed_control_out>255)
   {
    speed_control_out=255;
   }
   else if(speed_control_out<-255)
   {
    speed_control_out=-255;
   }

  //go forward
  if(speed_target>0)
  {
    float control_out=(255 + speed_control_out)/2;
    digitalWrite(M_CTRL_F, HIGH);
    digitalWrite(M_CTRL_B, LOW);
    analogWrite(SPEED_CONTROL_PIN,abs(int(control_out)));
  }
  //go backward
  else
  {
    float control_out=(255 - speed_control_out)/2;
    digitalWrite(M_CTRL_F, LOW);
    digitalWrite(M_CTRL_B, HIGH);
    analogWrite(SPEED_CONTROL_PIN,abs(int(control_out)));
  }
}

void run_motor()
{
  if(control_out>255)
  {
    control_out=255;
  }
  else if(control_out<-255)
  {
    control_out=-255;
  }

  if(control_out<0)
  {
    //go reverse
    digitalWrite(M_CTRL_B, HIGH);
    digitalWrite(M_CTRL_F, LOW);
    analogWrite(SPEED_CONTROL_PIN,abs(int(control_out)));

//    Serial.print("\nError:");
//    Serial.print(error_pos);
//    Serial.print("\nTargetpos:");
//    Serial.print(target_pos);
//    Serial.print("\ncurrentpos:");
//    Serial.print(current_pos);
//    Serial.println();
  }
  else
  {
    //go forward
    digitalWrite(M_CTRL_F, HIGH);
    digitalWrite(M_CTRL_B, LOW);
    analogWrite(SPEED_CONTROL_PIN,abs(int(control_out)));

//    Serial.print("\nError:");
//    Serial.print(error_pos);
//    Serial.print("\nTargetpos:");
//    Serial.print(target_pos);
//    Serial.print("\ncurrentpos:");
//    Serial.print(current_pos);
//    Serial.println();
  }
}

/*
 * Hari's code
 */
void sort(float a[])
{
    for(int i=0; i<25; i++)
    {
        bool flag = true;
        for(int j=0; j<(25-(i+1)); j++)
        {
            if(a[j] > a[j+1])
            {
                int t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                flag = false;
            }
        }
        if (flag)
        break;
    }
    //Serial.println(1);
}
/*********************/

void  send_telemetry()
{
  int M=0,D=0,S=0;
  switch(state)
  {
    case STATE0:
        M=0;
        D=0;
        S=0;
        break;
    case STATE1:
        M=current_pos;
        D=filtered_ultra;
        S=1;
        break;
    case STATE2:
        M=speed_actual;
        D=(int)(100.0*potVolt);
        S=2;
        break;
    case STATE3:
        M=servo_target;
        D=IRdistance;
        S=3;
        break;
    case STATE4:
        M=current_angle;
        D=fsrForce;
        S=4;
        break;
  }

  if(M>=500)
  {
    M=500;
  }
  else if(M<0)
  {
    M=0;
  }

  if(D>=500)
  {
    D=500;
  }
  else if(D<0)
  {
    D=0;
  }

  String string1 = "S";                                   // using a constant String
  String string2 =  String(S);
  String string3 = "M";                                   // using a constant String
  String string4 =  String(M);
  String string5 = "D";                                   // using a constant String
  String string6 =  String(D);
  String string7 =  "#";

  String string=string1 + string2 + string3 + string4 + string5 + string6 + string7;
  Serial.println(string);
}

void setup()
{
    Filter.begin();
    Filter_IR.begin();

    /*
     * Nick's code
     */
    myStepper.setSpeed(60);
    /**********************/

    /*
     * Hari's code
     */
     // put your setup code here, to run once:
     // put your setup code here, to run once:
      myservo.attach(SERVO);
      //myservo.write(0);
      pinMode(ir, INPUT);
      /***************************/

    /*
     * Ultrasonic sensor related stuff
     */
     pinMode(ultraPin, INPUT);

     /*
      * Motor initializations
      */
    pinMode(M_CTRL_B, OUTPUT);
    pinMode(M_CTRL_F, OUTPUT);
    pinMode(SPEED_CONTROL_PIN, OUTPUT);

    //initialize serial port
    Serial.begin(BAUD);

    //initialize input output pins
    pinMode(statusLedPin, OUTPUT);          // sets the digital pin 13 as output

    pinMode(button0_pin, INPUT);      // sets the digital pin 2 as input
    pinMode(ENCODER1, INPUT);      // sets the digital pin 3 as input
    pinMode(ENCODER2, INPUT);      // sets the digital pin 2 as input

    attachInterrupt(digitalPinToInterrupt(button0_pin), button0_pressed, RISING);
    attachInterrupt(digitalPinToInterrupt(ENCODER1), encoder1_change, CHANGE);

    Serial.print("State");
    Serial.print(state);
    Serial.println("\r");
}

void loop()
{
    //setting loop flag at 5Hz
    long time_now=millis();
    if((time_now-loop_last_time)>20)
    {
      loop_run_flag=1;
      loop_last_time=time_now;
    }

    bool val_button0=digitalRead(button0_pin);   // read the input pin
```

```
time_now=millis();
if(val_button0==true)
{
  if(button0_rise_detected==1)
  {
    if(time_now>button0_press_start_time)
    {
       button0_press_time=time_now-button0_press_start_time;
    }

    if(button0_press_time>DEBOUNCE_DELAY)
    {
      change_state();
      button0_press_time=0;
      button0_press_start_time=time_now;
      button0_rise_detected=0;
    }
  }
}

while(Serial.available())
{
     str= Serial.readString();// read the incoming data as string
     start_parse=1;
     char ch=str[0];
     if(ch=='D' || ch=='p' || ch=='i' || ch=='s' || ch=='x' || ch=='X' || ch=='S' || ch=='g')
     {
        str2=str;
        str2.replace(ch,'0');
        int val=str2.toInt();

        int test_bad=0;
        int i=1;
        while(str2.charAt(i)!='\0')
        {
            if(str2.charAt(i)<48 || str2.charAt(i)>57)
            {
              test_bad=1;
              break;
            }
            i++;
        }

         if(val>=0 && val<=360 && str.length()<=4 && test_bad==0)
        {
          int out=val;
          switch(ch)
          {
            case 'D':
```

```
Serial.print("\nReceived target:");
Serial.println(out);
 if(state==STATE1)
{
  target_pos=out;
  control_input=1;
  integrator=0;
}
else if(state==STATE2)
{

}
else if(state==STATE3)
{
  servo_target=out;
  control_input=1;
}
else if(state==STATE4)
{
  desired_angle=out;
  control_input=1;

  if(desired_angle==0)
  {
    stepper_init=1;
  }

}
break;
case 'p':
p_gain=(float)out;
Serial.print("\nPgain:");
Serial.print(p_gain);
break;
case 'i':
i_gain=(float)out;
Serial.print("\nIgain:");
Serial.println(i_gain);
break;
case 'g':
d_gain=(float)out;
Serial.print("\nDgain:");
Serial.print(d_gain);
break;
case 'S':

Serial.print("\nST:");
Serial.print(speed_target);
```

```
             if(state==STATE1)
            {
            }
            else if(state==STATE2)
            {
              speed_target=(float)out;
              control_input=1;
            }
            else if(state==STATE3)
            {
            }
            break;
            case 'x':
            control_input=0;
            break;
            case 'X':
            control_input=0;
            break;
          }
        }
      }
}

if(loop_run_flag)
{
  //clear the flag
  loop_run_flag=0;

  //Blink the status LED
  if(a==0)
  {
    a=1;
    digitalWrite(statusLedPin, HIGH);
  }
  else
  {
    a=0;
    digitalWrite(statusLedPin, LOW);
  }

  /**********************************/
  /*
   * Get and print ultrasonic data
   */
  ultraPulse = pulseIn(ultraPin, HIGH);

  //147uS per inch
  ultraInches = ultraPulse / 147;
  //change Inches to centimetres and add a low pass filter
```

```
      ultraCM = ultraInches * 2.54;

      if(ultraCM>20)
      {
        filtered_ultra= Filter.run(ultraCM);
      }

//      Serial.print(ultraCM);
//      Serial.print("ucm");
//      Serial.println();
//      Serial.print("fcm");
//      Serial.println(filtered_ultra);
      /*********************************/


      /*********************************/

      long int time_now=millis();
      float dt=(time_now-pid_old_time)/1000.0;
      pid_old_time=time_now;
      if(dt<0.01)
      {
        dt=0.01;
      }
      else if(dt>1)
      {
        dt=1;
      }

      //degree per sec
      speed_actual=(encoder_count-last_encoder_count)/dt;
      last_encoder_count=encoder_count;
//      Serial.print("Speed:");
//      Serial.println(speed_actual);

      send_telemetry();

      switch(state)
      {
        //Motors off
        case STATE0:

            //SHUT MOTORS
            stop_motor();

            //TODO: Add code for stopping servo and stepper motor

            //clear some variables
            encoder_count=0;
```

```
    last_encoder_count=0;
    target_pos=0;
    integrator=0;
    speed_integrator=0;
    speed_target=0;
    break;

  //DC motor pos control + ultrasonic sensor
  case STATE1:
    /*
     * PID control
     */
    if(control_input==0)
    {
      target_pos=map(filtered_ultra,20,200,0,360);
    }

    current_pos=encoder_count;
    error_pos=target_pos-current_pos;

    if(control_out<255 && error_pos>0)
    {
      integrator=integrator+error_pos*dt;
    }
    else if(control_out>-255 && error_pos<0)
    {
      integrator=integrator+error_pos*dt;
    }

    if(i_gain>0.1)
    {
      if(integrator*i_gain>INT_MAX)
      {
        integrator=INT_MAX/i_gain;
      }
      else if(integrator*i_gain<-INT_MAX)
      {
        integrator=-INT_MAX/i_gain;
      }
    }

    d_component=d_gain*speed_actual;
    if(d_component>200)
    {
      d_component=200;
    }
    else if(d_component<-200)
    {
      d_component=-200;
```

```
            }

            control_out=p_gain*error_pos + i_gain*integrator -d_component;
            run_motor();

            speed_integrator=0;
            speed_target=0;
//            Serial.print("current_pos:");
//            Serial.println(current_pos);
//            Serial.print("control_out:");
//            Serial.println(control_out);
            break;

          //DC motor speed control + potentiometer
          case STATE2:
            /*
             * Potentiometer read
             */
              potValue=analogRead(potReadPin);
              potVolt=(float)potValue*5.0/1023.0;
              potNorm=potVolt/5.0;

//            Serial.print("\rPot:");
//            Serial.println(potNorm);
              if(control_input==0)
              {
                speed_target=100.0 + 300.0*potNorm;
              }
            /*********************************/



//            Serial.print("\rSpeed_actual:");
//            Serial.println(speed_actual);
//            Serial.print("Speed_target:");
//            Serial.println(speed_target);

            if(speed_target==0)
            {
              //SHUT MOTORS
               stop_motor();
               speed_integrator=0;
               speed_control_out=0;
            }
            else
            {
              speed_error = speed_target-speed_actual;

              if(speed_control_out<255 && speed_error>0)
              {
                speed_integrator = speed_integrator + speed_error*dt;
              }
              else if(speed_control_out>-255 && speed_error<0)
              {
                speed_integrator = speed_integrator + speed_error*dt;
              }

              if(speed_integrator>SPEED_INT_MAX)
              {
                speed_integrator=SPEED_INT_MAX;
              }
              else if(speed_integrator<-SPEED_INT_MAX)
              {
                speed_integrator=-SPEED_INT_MAX;
              }

              speed_control_out = speed_integrator*speed_I + speed_error*speed_P;
              run_motor_speed();
            }

            target_pos=0;
            integrator=0;
            break;

          //Servo motor + IR distance sensor
          case STATE3:

            // put your main code here, to run repeatedly:

//        for (int i=0; i<25; i++)
//        {
//            // Read analog value
//            IRinput[i] = analogRead(ir);
//        }
//        sort(IRinput);

            filtered_IR= Filter_IR.run(analogRead(ir));

            IRvolt = map(filtered_IR,0,1023,0,5000);
            IRvolt1 = IRvolt/1000.0;
            IRdistance = 23.4 * IRvolt1 * IRvolt1 -115.7*IRvolt1 + 156.2; // from transfer function
            tmppos = map(IRdistance,40.0,70.0,0.0,180.0);

            if(control_input==0)
            {
            if(IRvolt1 >=0.85 && IRvolt1<=2.5)
            {
                servo_target=tmppos;
//              Serial.print(IRdistance);
//              Serial.println(servo_target);
               myservo.write(servo_target);
              //range is 0.85v to 2.5v
              //distance range is 13.24cm to 74.75cm
              //working range is 40 to 71cm

            //delay for servo in site is 15ms
            }
            else
            {
              IRdistance=-100;
            }
          }
          else
          {
//            Serial.print("\nhere");
            myservo.write(servo_target);
          }

            //SHUT MOTORS
            stop_motor();

            //TODO: Add code for stopping servo and stepper motor

            //clear some variables
            encoder_count=0;
            target_pos=0;
            integrator=0;
            last_encoder_count=0;
            speed_integrator=0;
            speed_target=0;
            break;

        //Stepper motor + force sensor
        case STATE4:

            digitalWrite(DIR, HIGH);

          fsrReading = analogRead(fsrPin);

          // analog voltage reading ranges from about 0 to 1023 which maps to 0V to 5V (= 5000mV)
          fsrVoltage = map(fsrReading, 0, 1023, 0, 5000);

          // The voltage = Vcc * R / (R + FSR) where R = 10K and Vcc = 5V
          // so FSR = ((Vcc - V) * R) / V        yay math!
          fsrResistance = 5000 - fsrVoltage;     // fsrVoltage is in millivolts so 5V = 5000mV
          fsrResistance *= 10000;                // 10K resistor
          fsrResistance /= fsrVoltage;
```

```
      fsrConductance = 1000000;          // we measure in micromhos so
      fsrConductance /= fsrResistance;

      // Use the two FSR guide graphs to approximate the force
      if (fsrConductance <= 1000) {
        fsrForce = fsrConductance / 80;
    //      Serial.print("Force in Newtons: ");
    //      Serial.println(fsrForce);
      } else {
        fsrForce = fsrConductance - 1000;
        fsrForce /= 30;
    //      Serial.print("Force in Newtons: ");
    //      Serial.println(fsrForce);
      }

      int time_on = 500 + fsrForce * 10;
    //    Serial.println(time_on);

        if(control_input==0)
        {
          desired_angle = fsrForce*3;
          step_size = int (stepsPerRevolution * desired_angle / 360.0);
//           Serial.println(desired_angle);
//           Serial.println(step_size);
          myStepper.step(step_size);

          current_angle+=desired_angle;
        }
        else if(stepper_init==1)
        {
          stepper_init=0;
          step_size = int (stepsPerRevolution * (-current_angle) / 360.0);
          myStepper.step(step_size);
          current_angle=0;
        }
        else if(desired_angle>0)     //check = sign
        {
          step_size = int (stepsPerRevolution * (desired_angle-current_angle) / 360.0);
          myStepper.step(step_size);
          current_angle+=(int)(desired_angle-current_angle);
          desired_angle=0;
        }


        if(current_angle>360)
        {
          current_angle=(int)(current_angle)%360;
        }
```

```
//          Serial.print("Desired_angle:");
//          Serial.println(desired_angle);
//          Serial.print("Current Angle:");
//          Serial.println(current_angle);

          encoder_count=0;
          target_pos=0;
          integrator=0;
          last_encoder_count=0;
          speed_integrator=0;
          speed_target=0;
          break;
     }
       /*************************/
    }

//  10Hz loop
    static int loop_tele=0;
    if(loop_tele>=10)
    {
     loop_tele=0;

//     Serial.print("\nError:");
//     Serial.print(error_pos);
//     Serial.print("\nTargetpos:");
//     Serial.print(target_pos);
//     Serial.print("\ncurrentpos:");
//     Serial.print(current_pos);
//     Serial.println();

     send_telemetry();
    }
    else
    {
     loop_tele++;
    }


}


/**************************************End of Function section***************************/
```