SENSORS & MOTOR CONTROL LAB
ILR #1

Individual Lab Report #1 | October 14, 2016

SAMBUDDHA SARKAR

TEAM G

EXCALIBR

Huan-Yang Chang
Man-ning Chen
Sambuddha Sarkar
Siddharth Raina
Yiqing Cai

1. INDIVIDUAL PROGRESS

1.1 Overview

In the sensors and motor control lab, I played multiple roles namely as project manager, troubleshooter, integration specialist and in-house electronics consultant. I wanted to add some functionality to the sensors and motors being used together hence I designed specific housings of the sensor motor pairs. I was responsible for the following sensors and motors:

1. Sensors: Ultrasonic & Potentiometer.
2. Motors: Servo & Stepper.

```
/*BATMAN was here......*/
/*
SENSORS
Potentionmeter Out: A0
Force Sensor :      A1
Thermistor :        A2
Ultrasonic :        A5

MOTORS
Servo:              10
DC Motor -
  encoder ch.A      2
  encoder ch.B      3
  speed             11
  direction         12
  enable            13
Stepper Motor -
  step              8
  direction         9
  enable            7
MISC
Toggle Switch:      5
State Button:       4
Buzzer              6
*/
```

Fig. 1.1, Pin map for components.

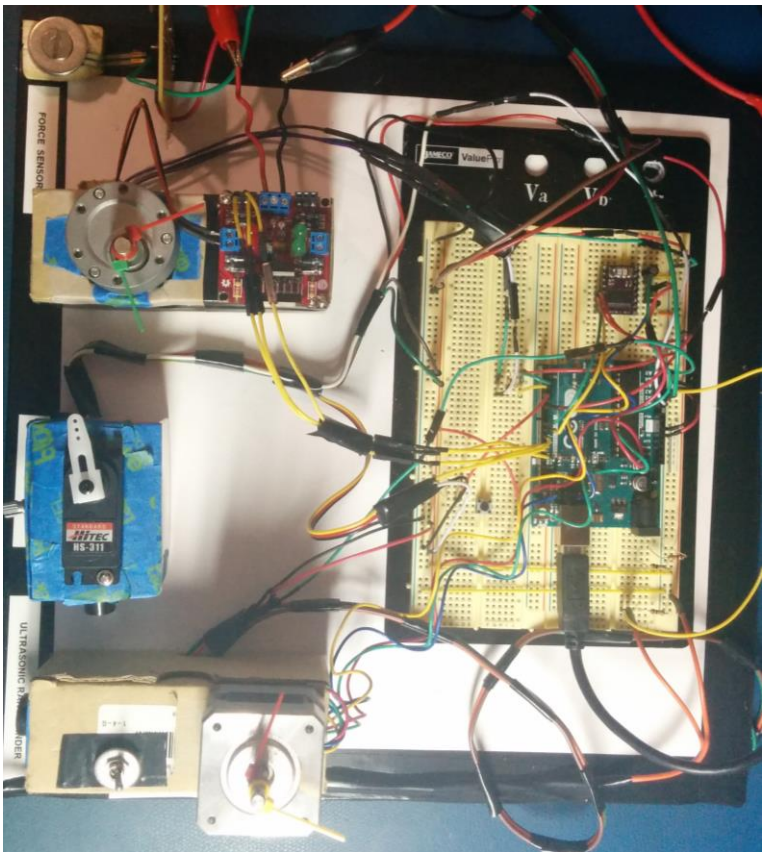


Fig. 1.2, Complete circuit.

1.2 Sensors

I have used 2 analog sensors for this lab. The potentiometer output was mapped to the servo input and the ultrasonic distance sensor's output was mapped to the stepper input. Fig. 1.2 shows the complete circuit and highlight the part of the circuit I was directly responsible for.

1.2.1 Potentiometer

The potentiometer is a passive sensor which is basically a voltage divider. The voltage is divided between 0-5V over a resistance range of 10K ohm. It was hooked up to the ADC of the Arduino UNO R3 on pin A0, the resolution of the ADC is 10-bit and as the knob is moved the voltage output varies between 0-5V with higher resistance equaling higher voltage. The Servo was operated between 0-180 degrees of rotation, so I scaled the readings of the potentiometer from 0-1023 to 0-180 numerically. The potentiometer location is highlighted in Fig. 1.2.1.

Fig. 1.2.1, Potentiometer

1.2.2 Ultrasonic Sensor

The ultrasonic sensor can be used in various modes and after playing around with the different modes, I chose to use it in analog mode as in my setup it produced more or less stable readings after applying a simple error reducing technique of averaging. (going against conventional wisdom which says PWM mode is better than ANALOG mode). I tested median and mode filters but it took too much memory as compared to the averaging filter and the accuracy of all the filters were pretty similar. One thing should be noted that these filters were software based i.e. no external hardware was used. The AN pin of the ultrasonic sensor was hooked up to the A5 of the Arduino UNO R3 input. The output was ranged between 0-5V over the 10-bit ADC (similar to Potentiometer in terms of the input). The sensitivity of the sensor was given as ~ 9.8mV/in, but after the transfer function was obtained by one of my team

mates, it was discovered that the sensitivity of the sensor in the setup was about ~9.1364mV/in which helped in improving our ultrasonic sensor performance. The output of the sensor was mapped to the stepper motor and buzzer. This was basically a mock-up of the reverse parking sensor in automobiles. As the distance of the object from the ultrasonic sensor reduced the speed of the stepper motor decreased and the frequency with which the buzzer beeped increased. So basically, the ultrasonic sensor output was mapped inversely to stepper motor rotation speed and directly to the time between subsequent beeps from the buzzer. It is important to note that the range of the ultrasonic sensor is from about 7 to 254 inches, meaning that for any obstacle placed between 0-7 inches from the sensor would give the same output. The ultrasonic sensor location is highlighted in Fig. 1.2.2.

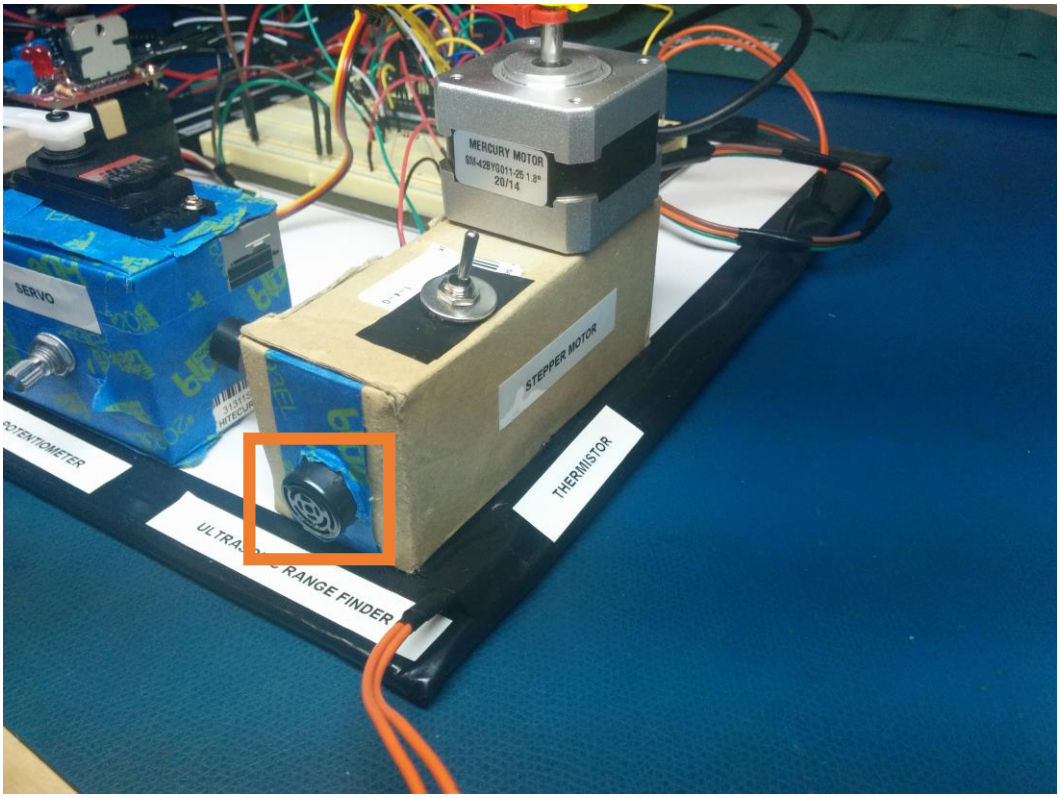


Fig. 1.2.2, Ultrasonic sensor

1.3 Motors

I was primarily responsible for two motors: Servo and Stepper. The servo was controlled using the built-in Arduino library but for the stepper rotation and speed control, I wrote my function.

1.3.1 Servo

The servo was input was mapped from the potentiometer output and the rotation is limited between 0-180. A simple knob control was implemented, in which when the potentiometer is rotated from the 0-10K ohm position the servo rotates from 0-180 degrees. The servo is controlled using PWM, so it was hooked up to the pin 10 (PWM) of Arduino UNO R3. Only position control was implemented. The servo location is highlighted in Fig. 1.3.1.

Fig. 1.3.1, Servo

1.3.2 Stepper

The stepper used in this assignment was a bi-polar stepper motor. The stepper motor was used in a full-step mode with each step being equal to 1.8 degrees. The bi-polar stepper is easier to control than an unipolar stepper motor. The stepper was interfaced with Arduino UNO R3 using a Pololu high current motor driver namely the DRV-8825. The pin mapping can be seen from Fig 1.3.2a given below.

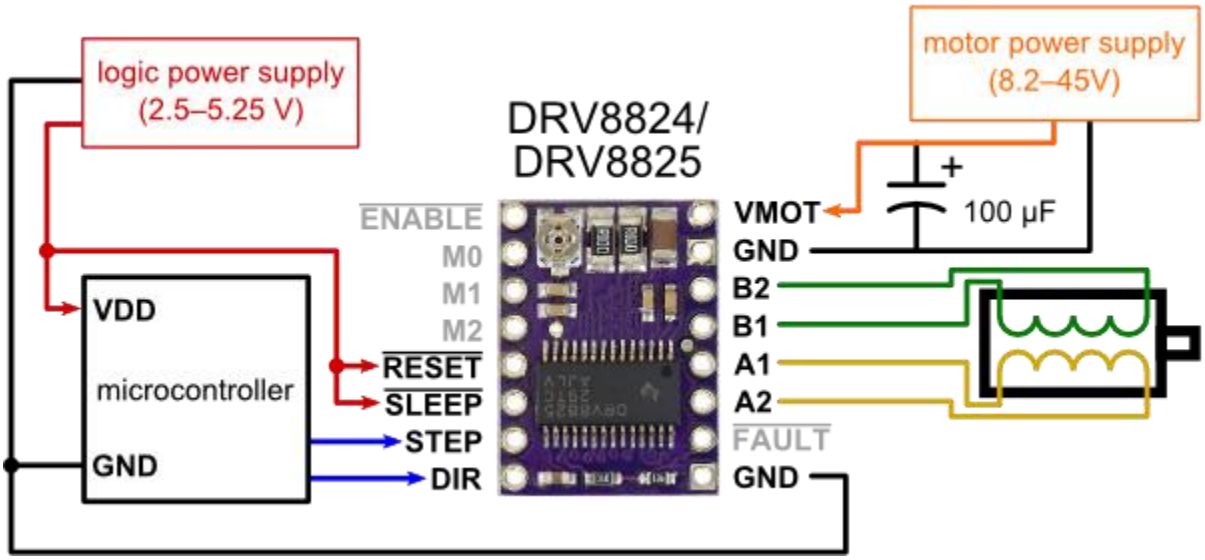


Fig. 1.3.2a, Stepper motor drive pin-out diagram and connection suggestion.

The driver pin out is shown in Fig. 1.3.2a. The driver has 4 output pins for the coils of the stepper and can be hooked up directly to the driver pins named B1, B2, A1 & A2. The stepper driver has 3 inputs from the microcontroller namely the Enable pin(ENABLE), Step pin(STEP) and Direction pin(DIR) which is hooked up to the pins 7,8 & 9 respectively of the Arduino UNO R3. The enable pin helps switch the motor ON/OFF thus reducing the current draw during idle time and prevent heating up of the motor. The motor was observed to heat up a lot when left idle with the enable pin on. The enable pin is operated by inverse logic, that is when it is high the enable pin is disabled and when low, the enable pin is enabled. The step input of the driver accepts the number of steps (which directly governs the rotation of the motor). The direction pin controls the direction of rotation, when it is given a logic high input it rotates in clockwise and for logic low its rotates in an anti-clockwise direction.

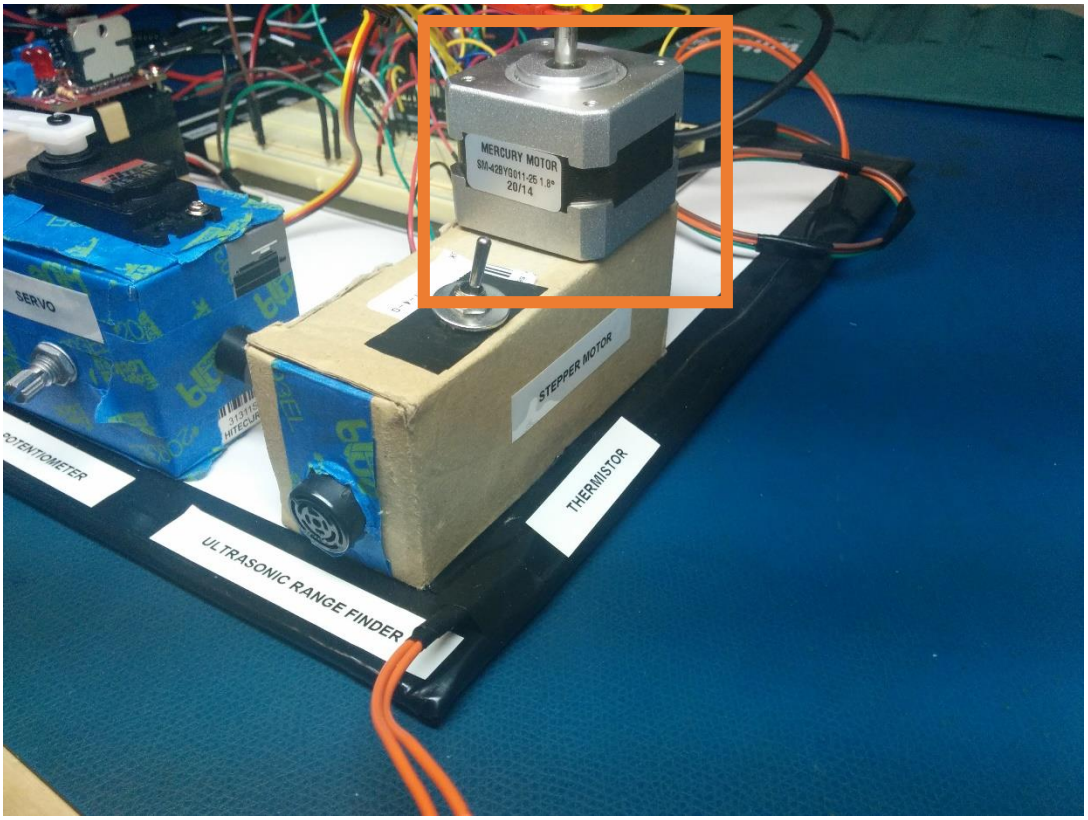
Position control of stepper: The position of the stepper can be controlled by directly giving the number of steps. Here as in full-step mode each step rotates the motor shaft by 1.8 degree, the total steps required for complete 360 rotation is 200. The number of steps required for a desired angular rotation can be calculated just by dividing the required rotation in degrees by 1.8.

Speed control of stepper: The speed can be controlled by simply increasing the time taken between consecutive steps. After a bit of number crunching it can be derived that:

RPM = $\frac{300}{t}$; where ‘t’ is the time in milliseconds between each consecutive step.

Hence the RPM can be given as an input to the stepper function and it will automatically calculate the time between consecutive steps.

A custom stepper function was created to operate stepper in full-step mode. It is important to note that the stepper motor driver's output current setting is a bit temperamental and for the given motor the V_{ref} was set to 0.250 V. This helps us limit the current in each coil of the stepper motor to 0.3A (rated current per coil is 0.33A). The stepper location is highlighted in Fig. 1.3.2b.



Fig, 1.3.2b, Stepper motor.

1.4 Miscellaneous

I used a simple push button switch to implement a state machine and buzzer to give a sound output to signal proximity, (when paired with ultrasonic sensor).

1.4.1 Buzzer

A simple piezo buzzer was hooked up with the Arduino UNO R3 on pin 6. The buzzer was simply added to give the ultrasonic sensor output an audio-visual appeal. The buzzer beeping frequency was directly related to the distance from the ultrasonic sensor, so as the obstacle got

closer the beeping frequency increase. A simple delay between the logic HIGH and LOW output on pin 6 did the trick. The piezo buzzer location has been highlighted in Fig. 1.4.1.

Fig. 1.4.1, Piezo buzzer.

1.4.2 Push button

The push button from the RGB LED assignment was carried over (with a software debounce) to this circuit. The purpose of the push button was to cycle through different states of the circuit operation. The default state was the GUI input mode; the other states are as follows:

- State 1: Potentiometer output mapped to the Servo.
- State 2: Ultrasonic sensor output mapped to Stepper motor and piezo buzzer.
- State 3: Force sensor output to the Servo.
- State 4: Thermistor output to the DC motor.

The push button was connected to pin 4 on the Arduino UNO R3. Fig. 1.4.2 highlights the location of the push button.

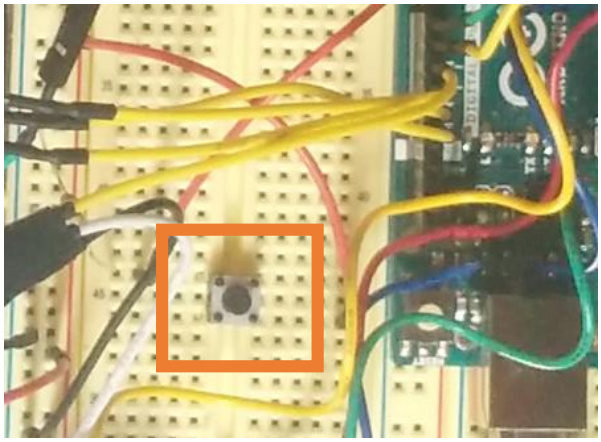


Fig. 1.4.2, Push button.

1.5 Integration

The integration was performed in 2 stages; first, the hardware components were integrated then the Arduino code for different components were integrated into one seamless code with multiple functions.

1.5.1 Hardware

The sensors and motors had been tested and operated by different team members, so the first task at hand was to develop a layout to accommodate all the sensors and motors. The layout was designed to prevent any conflicts between components of the project while drawing power from the Arduino or the independent power supply. Also, the signal wires had to be arranged as to avoid any entanglement as this could lead to faulty connections by creating unnecessary noise (due to various unshielded signal wires overlapping) and cross connections in the circuit.

The individual components where then grouped together according to the functionality of the circuit; for example, the servo and potentiometer were housed in a single box. This organization of sensor and motor pairs helped to debug the circuit and also eased the process of connecting the setup to Arduino. The cables were color coded to aid the process of connecting the sensors and motors.

The sensors and motors were tested individually by the team members on the bread board, hence it was important to have uniformity while mounting the sensor and motor circuits in the final setup. This was achieved by namely; color coding the cables, soldering the sensors to the said cables with headers and applying heat shrink tubing to any exposed leads or header-cable solders.

1.5.1 Software

The software integration turned out to be less of a hurdle than expected. As the team members were directed to stick to a format for writing codes for their sensors and motors, it definitely helped to ease the task of integrating the codes. Though I had to create custom functions for each motor which could be called at different instances of the code as needed.

The memory taken up by variables were too large at one point and I combed through the codes and assigned local variables where-ever possible. The different values of delay() used in different sensor and motors' code did conflict with the logic flow of the program sometimes but it was rectified by simply reducing the delay() value.

Finally, to switch the circuit between different operation modes, a state machine was programmed (using conditional statements) into the code which enabled easier overall operation and debugging of the code.

2. CHALLENGES

There were multiple challenges that I faced while working the stepper motor and the overall integration of the circuit. These have been listed below.

2.1 Faulty solder-less bread board

The bread board had some manufacturing defect which led to regular faults in the circuits, specially whilst operating the motors. This was rectified by porting the circuit to a new bread board. Sometimes, the pin holes in the bread board weren't properly molded, rendering those pins useless.

2.2 Stepper Motor & Stepper Driver

The current setting on the Pololu motor driver was a bit of a challenge, the potentiometer wasn't operating properly on our original driver and this lead to the blowing up of the electrolytic capacitor which lead to a chain of events which ultimately shorted the motor driver itself. A new motor driver was chosen and then interfaced with the Arduino once again. This rectified the problem and the current setting was achieved as per the motor rating by setting the V_{ref} voltage as 0.150 V.

The stepper motor has a tendency of heating up over time if left idle for a considerable time, this was addressed by switch the motor driver on and off using the ENABLE pin.

Though the above problems were solved, but during the final presentation the stepper motor drew too much power and caused a power surge leading to continuous resetting of the circuit. Due to this the code of the servo had to be commented.

2.3 Slaving Arduino UNO to GUI

We used processing to build the GUI and there were two ways of controlling the circuit with the GUI. One option was to slave the Arduino to the processing environment and the other one was to read data from the serial port of the Arduino. The former option is easier to integrate and we tried to do that but we faced a problem with using the interrupts while the Arduino was slaved to the processing environment. The DC motor's encoder used interrupts to count the pulses, but this interfered when the Arduino was slaved to the GUI. Hence we adopted the latter option, though the interfacing was bit of a problem, but we got around it and developed our own set of custom commands to help communicate between the Arduino and the GUI environment. The command list has been listed in Fig. 2.3.

```
/* COMMANDS CHEAT SHEET

(Motor)(Degree of Rotation)(Speed of Motor)/
OR
(Sensor)(Val of Sensor)/

A-  Servo
B-  Stepper Forward Speed
C-  Stepper Reverse Speed
D - DC Forward Speed
E - DC Reverse Speed
V - Stepper Forward Degree
X - Stepper Reverse Degree
Y - DC Forward Degree
Z - DC Reverse Degree

P-Potentionmeter
U-Ultrasonic
F-Force
T-Thermistor
 give space after character.
Val : 0-255

*/
```

Fig. 2.3, Command Set.

3. TEAM WORK

The project work was divided among the team members and the task was assigned according to the strengths of the team members. The task division has been listed below in Table 3.1.

Team Member	Task
Yiqing Cai	GUI.
Huan-Yang Chang	DC motor, PID Control, Ultrasonic sensor (PWM).
Man-ning Chen	Thermistor & Audio-potentiometer.
Siddharth Raina	Force Sensor, Ultrasonic Sensor Plot.
Sambuddha Sarkar	Stepper, Servo, Potentiometer, Ultrasonic (AN & PWM), Buzzer & Integration.

Table 3.1, Task Division.

4. FUTURE PLANS

My future plans until the next progress report is to get the AEROTECH robotics arm up and running at Oculus research. The main challenge is to get the AEROTECH arm operational is getting used to the command set for controlling the arm and also get used to the A3200 controller. The motion planning algorithm for the arm has to be developed such that it can position the calibration target at desired position with the least amount of jitter and in a timely manner.

The motion of the arm of the arm has to be synchronized with the camera trigger and this can only be done after the motion planning algorithm has been developed properly.

5. RAW CODE

```
/*BATMAN was here.....*/
/*
SENSORS
Potentiometer Out: A0
Force Sensor :     A1
Thermistor :       A2
Ultrasonic :       A5

MOTORS
Servo:            10
DC Motor -
 encoder ch.A     2
 encoder ch.B     3
 speed           11
 direction       12
 enable          13
Stepper Motor -
 step             8
 direction        9
 enable           7
MISC
Toggle Switch:    5
State Button:     4
Buzzer            6
*/

/* TOGGLE SWITCH is for switching between GUI & Sensor control */

/* COMMANDS CHEAT SHEET

(Motor)(Degree of Rotation)(Speed of Motor)/
OR
(Sensor)(Val of Sensor)/

A-  Servo
B-  Stepper Forward Speed
```

```
C-  Stepper Reverse Speed
D - DC Forward Speed
E - DC Reverse Speed
V - Stepper Forward Degree
X - Stepper Reverse Degree
Y - DC Forward Degree
Z - DC Reverse Degree

P-Potentionmeter
U-Ultrasonic
F-Force
T-Thermistor
 give space after character.
Val : 0-255

*/

#include <Servo.h>
#include <Timer.h>

Servo servo_1;
Timer t;

//pin allocation
//SENSORS
const int pot_knob = A0, f_sensor = A1, therm = A2, ul_sonic = A5, toggle_switch = 5;
//MOTOR
const int servo = 10; //Servo
const int ch_A = 2, ch_B = 3;// DC Encoder
const int dirPinB = 12, enablePinB = 13, speedPinB = 11; //DC Motor
const int stp_step = 8, stp_dir = 9, stp_enable = 7; // Stepper
//MISC.
const int state_chngr = 4, buzzer = 6;

//variables
int state = 0, bttn_state = 0, last_bttn_state = 0, bttn_counter = 0;
int pot_val = 0, servo_pos = 0;
int millisbetweenSteps, steps; // stepper speed, degrees
float stepper_rpm = 0;
int dist_inches = 0;
int mode =1;
//DC Motor

// rpm_parameter
int encoderValue_A = 0;
bool motor_clockwise_rpm; // GUI take this to show direction
bool motor_not_move = false;
float motor_rpm = 0; // GUI take this to show rpm
float resolution_per_circle = 180, time_period =1000, motor_deg;

// PID_rpm
long lastTime_PID;
double Input, Output,Setpoint;
double errSum, lastErr;
int errSum_count = 0;
double timeChange;
double  error , dErr;
```

```
double kp = 0.5;
double ki = 1;
double kd = 0;
// PID_degree
double kp_deg = 0.3;
double ki_deg = 0.2;
double kd_deg = 0;
bool model_rpm = true;
int degree; // the gui should set the degree here!
bool degree_dir;
int deg_setpoint; // the GUI should set degree here
int model_change;
int direction_change;
int Setpoint_change;



//timer_variable
int tickEvent;
int rpm_setpoint;// the GUI should set the rpm here!
bool set_motor_clockwise; // GUI shoul change the direction here ( true = clockwise, false = counter clockwise)
// everytime GUI try to change the  rpm direction  should stop the motor first (used stop_motor() and reset_PID())
// interrupt function
void count_rpm(void) {
 float aa = encoderValue_A;
 motor_rpm = aa/resolution_per_circle/time_period *1000*60;
 if (motor_rpm > 100){
   motor_rpm = 0;
 }
 encoderValue_A = 0;
}
void count_A() {
 encoderValue_A++;
   if(digitalRead(ch_B) == HIGH) {
     motor_clockwise_rpm  = false;
     degree = degree -2;// because the encoder 180 puise per circle
     }
   else {
   motor_clockwise_rpm  = true;
   degree = degree +2;
     }
}
void Compute(int z){
 // load input
  /*How long since we last calculated*/
  unsigned long now = millis();
  double timeChange = (double)(now - lastTime_PID) / 1000;

  /*Compute all the working error variables*/
   error = Setpoint - Input;
   errSum += (error * timeChange);
   dErr = (error - lastErr) / timeChange;

  /*Compute PID Output*/
  //pid_parameter_rpm
  if (z ==1){
  Output = kp * error + ki * errSum + kd * dErr;
```

```
  }
  //pid_parameter_degree
  if (z == 0){
  Output = kp_deg * error + ki_deg * errSum + kd_deg * dErr;
  }

  /*Remember some variables for next time*/
  lastErr = error;
  lastTime_PID = now;
}

void setup()
{
 pinMode(ul_sonic, INPUT), pinMode(servo, OUTPUT),pinMode(buzzer, OUTPUT), pinMode(state_chngr, INPUT), pinMode(toggle_switch,
INPUT);
 //DC & Stepper
 pinMode(pot_knob, INPUT), pinMode(f_sensor, INPUT), pinMode(therm, INPUT);
 pinMode(state_chngr, INPUT);
 //Define L298N Dual H-Bridge Motor Controller Pins
 pinMode(dirPinB,OUTPUT);
 pinMode(enablePinB,OUTPUT);
 pinMode(speedPinB,OUTPUT);
 pinMode(stp_step, OUTPUT), pinMode(stp_dir, OUTPUT);

 // encoder pin channel A = 2(interrupt) , channel B = 3(digital_read)
 pinMode(ch_A, INPUT_PULLUP);
 pinMode(ch_B, INPUT_PULLUP);
 attachInterrupt(digitalPinToInterrupt(ch_A),count_A,RISING);
 // rpm_setup
 tickEvent = t.every(time_period, count_rpm);
 servo_1.attach(10);

 Serial.begin(9600);//Serial Comm.
 Serial.flush();


}

void loop()
{
 t.update(); // let timer work
 //state =1;
 //if (analogRead(toggle_switch)== HIGH)
   mode = 1;
   if(mode ==1)
   {
   //Serial.println("Sensor Contorl");
    {
      bttn_state = digitalRead(state_chngr);
     if (bttn_state != last_bttn_state)
      {
        if (bttn_state == HIGH)
          {
           // HIGH = on
            bttn_counter++;
            state = (bttn_counter % 5);
            //Serial.print( "STATE : ");
```

```
            //Serial.println(state);
           }
         delay(100);//debounce
         last_bttn_state = bttn_state;
      }

   if (state == 1) //Servo with Potentionmeter : 0-10k ohm mapped to 0-180 deg.
    {

      int servo_deg = map(analogRead(pot_knob), 0, 1023, 1, 179);
      servo_1.write(servo_deg);
      //Serial.println(servo_deg);
      Serial.print("A");Serial.print(servo_deg);Serial.print(" ");
      Serial.print("P");Serial.print(analogRead(pot_knob));Serial.print(" ");
      delay(50);
     //Serial.print( "STATE : ");
     //Serial.println(state);



    }

   else if(state == 2) //Stepper with Ultrasonic : Distance inversely proportional to buzzer beep frequency & directly proportional to motor
speed
    {


      //Ultrasonic Sensor
      //filter_ul_sonic ((analogRead(ul_sonic)/2));
      dist_inches = (analogRead(ul_sonic)/2);
      stepper_rpm = (map(dist_inches, 6, 100, 0, 200)); //A 5V supply yields ~9.8mV/in, the value has to be divided by 2 to get the actual inches
      Serial.print("B");Serial.print(stepper_rpm);Serial.print(" ");
      Serial.print("U");Serial.print(dist_inches);Serial.print(" ");
      //Stepper Control
      //stepper(stepper_rpm,9000,1);
      //Serial.println(dist_inches);
      delay(50);
      //Serial.print( "STATE : ");
      //Serial.println(state);
      }

   else if(state == 3) //Servo with Force sensor (rotary) : 0-7 N mapped to 0-180 deg.
    {
      int servo_deg = map(analogRead(f_sensor), 0, 300 , 0 , 179);
      servo_1.write(servo_deg);
      Serial.print("A");Serial.print(servo_deg);Serial.print(" ");
      Serial.print("F");Serial.print(analogRead(f_sensor));Serial.print(" ");
      //Serial.println(servo_deg);
      delay(50);
      //Serial.print( "STATE : ");
      //Serial.println(state);
    }

   else if(state == 4) // DC with Thermistor : DC fan speed increse with temperature rise, after threshold buzzer warning.
    {
      double thermometerReading = analogRead(therm);
      double resistance = 10000.0*(1023.0-thermometerReading)/thermometerReading;
```

```
      if(resistance>8059)//30 degree
        {//"Temperature is too low";
          //DC motor doesn't work
          dc_motor(0, 1, 1);
          Serial.print("D");Serial.print(0);Serial.print(" ");
          Serial.print("T");Serial.print(analogRead(therm));Serial.print(" ");
          //Serial.println("zero speed");
        }
      else if(resistance<3605){//50 degree
      //"Temperature is too high";
          //DC motor to full speed
          dc_motor(60, 1, 1);
          Serial.print("D");Serial.print(motor_rpm);Serial.print(" ");;
          Serial.print("T");Serial.print(analogRead(therm));Serial.print(" ");
          //Serial.println("half speed");
        }
      else{
      //"Safe temperature";
          //DC motor half speed
          dc_motor(30, 1, 1);
          Serial.print("D");Serial.print(motor_rpm);Serial.print(" ");
          Serial.print("T");Serial.print(analogRead(therm));Serial.print(" ");
          //Serial.println("half speed");
          }
    delay(100);
    //Serial.print( "STATE : ");
     //Serial.println(state);
    }
else if(state == 0)
{
// Serial.println("[[[[[[GUIQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQGUI");
//GUI Contorl
 //Serial.println("GUI Contorl");
 if (Serial.available() > 0) {
  String raw_read = Serial.readString();
  if (raw_read[0] == 'Y') {
    model_change  = 0;
    direction_change  = 1;
    Setpoint_change = raw_read.substring(1).toInt();
    degree = 0;
  }
  else if (raw_read[0] == 'D') {
    model_change  = 1;
    direction_change  = 1;
    Setpoint_change = raw_read.substring(1).toInt();
    degree = 0;

  }
  else if (raw_read[0] == 'E') {
    model_change  = 1;
    direction_change  = 0;
    Setpoint_change = raw_read.substring(1).toInt();
    degree = 0;

  }
  else if (raw_read[0] == 'Z') {
    model_change  = 0;
```

```
    direction_change  = 0;  ;
    Setpoint_change = raw_read.substring(1).toInt();
    degree = 0;
   }
  }
  //Serial.println("EEEEE");
  //Serial.println(Setpoint_change);
  //Serial.println(direction_change);
  //Serial.println(model_change);
  //Serial.print(" ");
  dc_motor(Setpoint_change,direction_change,model_change); // Dc_motor
  if( model_change ==0) {
   if(direction_change ==0) {
    dc_motor((-Setpoint_change),direction_change,model_change); // Dc_motor in negative degree
    //Serial.println("QQQQQ");
   }

  }

  if (motor_clockwise_rpm == true) {
  Serial.print("D");Serial.print(motor_rpm);Serial.print(" ");

  }
  else {
   Serial.print("E");Serial.print(motor_rpm);Serial.print(" ");
  }
  if (direction_change == 0){
  Serial.print("Z");Serial.print(abs(degree));Serial.print(" ");
  }
  else{
  Serial.print("Y");Serial.print(abs(degree));Serial.print(" ");
  }
   }
   }

 }

 }

int filter_ul_sonic (int x)
{
 int sum = 0; //Create sum variable so it can be averaged
 int avgrange = 10; //Quantity of values to average (sample size)
  for (int i = 0; i < avgrange ; i++)
   {
    sum += x;
    delay(10);
  }
 // offset 1 inch by exeriemnt
 dist_inches = (sum / avgrange)+1;
 sum = 0;
 return dist_inches ;
}


//functions for Stepper & DC Motor;
```

```
// Stepper Motor Function
int stepper(int x, int y, int z) //speed, angle, direction
 {


  if(z == 1)//Forwward
  {
    digitalWrite(stp_dir, HIGH);
    for(int n = 0; n < y; n++)
     {
       digitalWrite(stp_step, HIGH);
       digitalWrite(stp_step, LOW);
       delay(300/x);
     }
  }

  else if(z == 0)//Reverse
  {
    digitalWrite(stp_dir, LOW);
    for(int n = 0; n < y; n++)
     {
       digitalWrite(stp_step, HIGH);
       digitalWrite(stp_step, LOW);
       delay(300/x);
     }
  }
 }


// DC Motor Function
int dc_motor(int x, int z, int w)// speed, angle, direction(+1: Clockwise, 0: counter), mode(+1: rpm, 0: degree)
{
  Compute(z);
if( w == 1) {
// the  GUI should delete this line, just for arduino to use read data to control
 rpm_setpoint = x;
//delete end
Setpoint =rpm_setpoint;
Input = motor_rpm;
int value;
  if (z == 1){
   value = 255-Output-50; // -50 only for offset motor's voltage to a better place for quick start
   if (value <0) {
     value = 0;
   }
   else if (value >255){
     value = 255;
   }
   analogWrite(speedPinB, value);
   digitalWrite(dirPinB, HIGH);
   digitalWrite(enablePinB, HIGH);
   }
 else if(z == 0)
 {
   value = Output+50;// +50 only for offset motor's voltage to a better place for quick start
     if (value <0) {
```

```
      value = 0;
    }
    else if (value >255){
      value = 255;
    }
    analogWrite(speedPinB, value);
    digitalWrite(dirPinB, LOW);
    digitalWrite(enablePinB, HIGH);
   }
}
if( w == 0) {
// the  GUI should delete this line, just for arduino to use read data to control
deg_setpoint = x;
//delete end
Setpoint = deg_setpoint;
Input= degree;
int value;
 if (Setpoint >Input){
   value = 255-Output-50;// -50 only for offset motor's voltage to a better place for quick start
   if (value <0) {
     value = 0;
   }
   else if (value >255){
     value = 255;
   }
   analogWrite(speedPinB, value);
   digitalWrite(dirPinB, HIGH);
   digitalWrite(enablePinB, HIGH);
   }
 else if(Setpoint <Input)
 {
   value = Output+30;// +50 only for offset motor's voltage to a better place for quick start
     if (value <0) {
       value = 0;
     }
     else if (value >255){
       value = 255;
     }
     analogWrite(speedPinB, value);
     digitalWrite(dirPinB, LOW);
     digitalWrite(enablePinB, HIGH);
   }
   else if (Setpoint == Input){
     analogWrite(speedPinB, 0);
     digitalWrite(dirPinB, LOW);
     digitalWrite(enablePinB, LOW);
   }
   // in degree model don't let motor rotate too much, set a time to stop
   delay(50);
   analogWrite(speedPinB, 0);
   digitalWrite(dirPinB, LOW);
   digitalWrite(enablePinB, LOW);
}


}
```

```
// PID function  mode(+1: rpm 0: degree)



void reset_PID() {
  lastTime_PID = millis();
  errSum, lastErr= 0 ,0;
  error , dErr= 0 ,0;
}

void stop_motor() {
   analogWrite(speedPinB, 0);
   digitalWrite(dirPinB, LOW);
   digitalWrite(enablePinB, LOW);
   delay(200);
}


// reset_all
void reset_all() {
   stop_motor();
   reset_PID();
   delay(100);// let motor really stop
   Output = 0;
   encoderValue_A = 0;
   degree = 0;
}
```