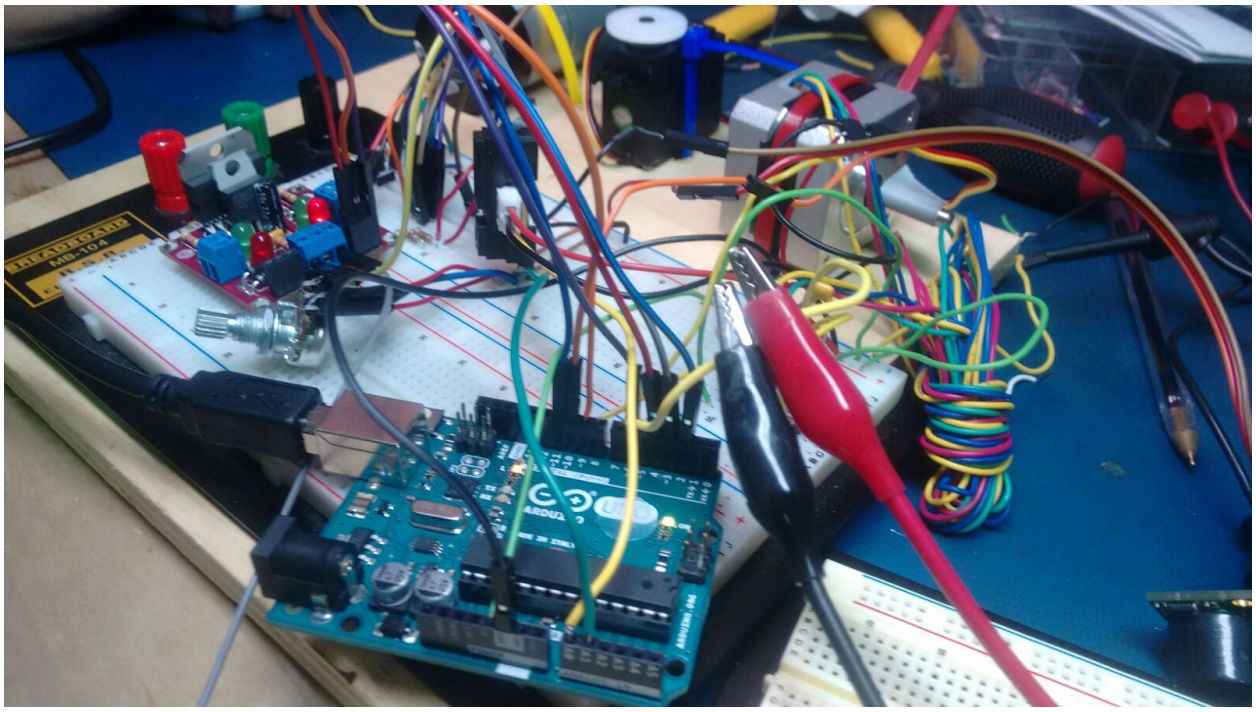
INDIVIDUAL LAB REPORT
Sensors and Motor Control
MRSD Project

Name: Keerthana P G
Andrew ID: kgopalak

Team D



Responsibility

I was responsible for devising how to do debouncing, speed control of DC motor using potentiometer as sensor and PID control of DC motor via GUI input. I also transferred all the circuits to one board in hardware integration.

Part 1: Debouncing using Push Button Switch

What is debouncing? Why is it important?

Pushbutton[1]s generate spurious open/close transitions when pressed which may be read as multiple presses in a very short time. Debouncing is the act of measuring input only once within a fixed short period of time. Without debouncing, pressing the button causes unpredictable results.

Fig 1: Standard push button switch[2]

Here, the push button was connected to an interrupt pin which read HIGH/LOW once every 0.05 seconds, depending on when it's pressed. This was used to switch between different states, such as GUI/sensor control.

Pseudo code:

```
loop{
reading = digitalRead(pin);
if(reading != previous1)
  {
    if (reading == HIGH)
          changeState();
    delay(50);
  }
  previous1=reading1;
}
```

Part 2: DC Motor Velocity Control using Potentiometer

The DC motor converts electrical energy into mechanical energy by force on a current carrying arm in a magnetic field. We used the SPG30-60K motor[3] with mounted encoder shown in Figure 2. It has a rated voltage of 12V and rated speed of 12RPM.

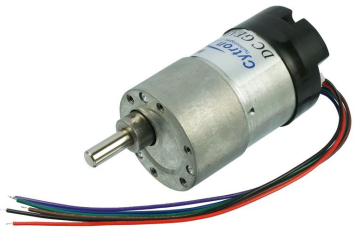


Figure 2: DC motor with encoder[3]

We used L298 motor driver, shown in figure 3, to control the DC motor.

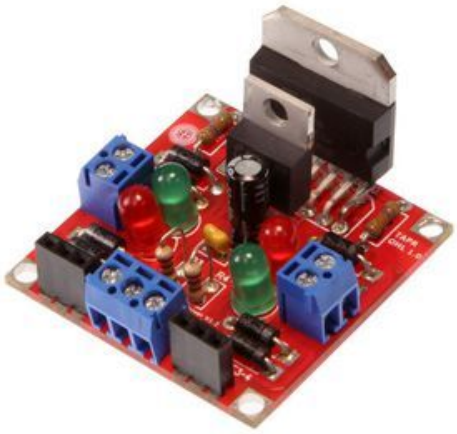


Figure 3: L298 Compact Motor Driver[4]

The logic was controlled by two motor driver pins(L1,L2) and an Enable pin.

E	L1	L2	Result	E	L1	L2	Result
0	0	0	Brake	1	0	0	Brake
0	0	1	Brake	1	0	1	Forward
0	1	0	Brake	1	1	0	Backward
0	1	1	Brake	1	1	1	Brake

We used a rotary potentiometer to control motor.

Figure 4: Rotary Potentiometer[5]

At its core, it is a three terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. When connected across a voltage(5V here) it read a value between 0 and 1023 when turned. This value was suppressed within 0-255 and written into the Enable pin as a PWM wave. Thereby, turning the motor alters voltage across motor and changes speed. *The transfer function of this sensor is linear.*

Part 3: DC Motor Position Control using PID

The PID control algorithm responds to proportional, integral and differential error.

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(\tau)\,d\tau + K_\mathrm{d} \frac{de(t)}{dt},$$

The respective coefficients are non-negative.

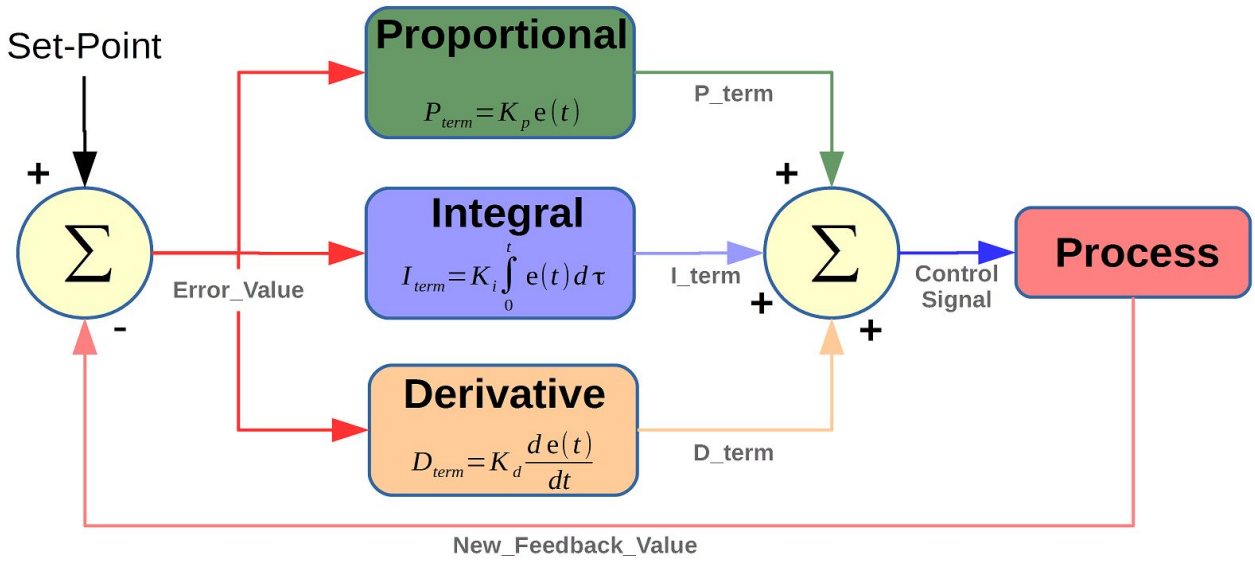


Figure 5: Schematic of PID Control Algorithm[6]

A typical response curve looks like Figure 6

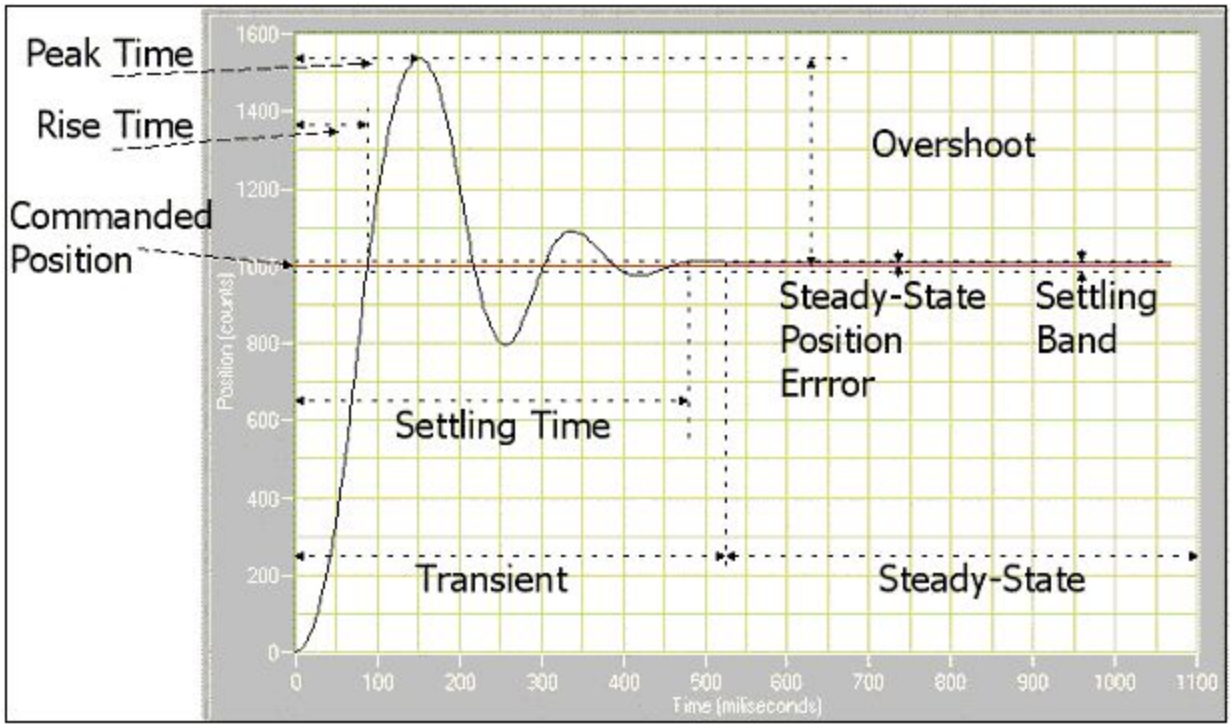


Figure 6: Response Curve of PID control

Here, *settling time* is determined by Kp, *steady state oscillations* by Kd and *offset* at steady state by Ki. By regulating three values one can obtain a desired response curve.

On rotary encoders

Here, error is given by a rotary encoder, that converts angular position to digital signal. One of the encoder pins was connected to an interrupt pin on Arduino to check for position change. At the interrupt, the two pins are read, and if their values differ, it indicates a change in position. This change is tracked to get measure the actual position of the motor and sent as feedback to the network.

Interesting Observations and Inferences

1. It was found that for low PWM cycles, the DC motor wouldn't move. If PWM value was less than 116, the motor stood stand still. At 116, there is an abrupt change in voltage across the motor. This was a non linearity in motor behaviour. When motor was still at non zero PWM voltage, one could observe a high pitched sound that increases with voltage that perhaps dissipated power and signified interference with other armatures/fields on board.
2. The encoder caused too much noise when not used in combination with a pull up resistor. Therefore it is advised to pull up both the encoder pins.
3. At high velocities(PWM) the DC motor stabilised with a lot of jerks, this was caused by overshooting desired value.

4. While the PID control worked perfectly on its own, when integrated with circuit, it continuously oscillated without stabilising. We figured this happened due to excessive loop time, ie, since the PWM was updated at every iteration of the loop, when the loop takes long to execute, the motor has already overshot the desired value. This continuous for subsequent updates, putting the motor in a steady state large oscillation. To tackle this, we optimized code to prevent unnecessary if and for conditions, and applied timeout to functions like Serial.readString(). Lower the timeout, better was stabilisation.
5. While doing hardware integration involving components designed by others, one should always take pictures before removing wires/moving components.

References

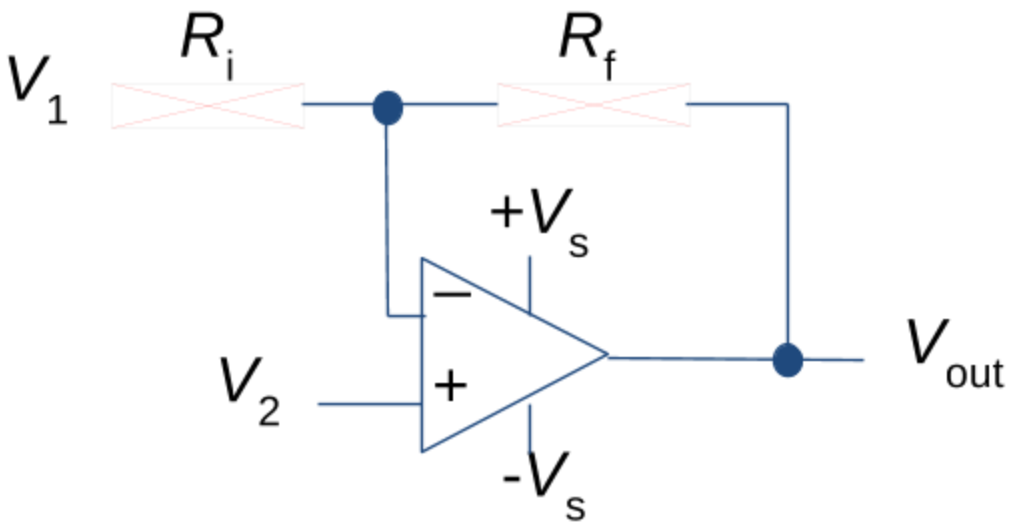
1. https://www.arduino.cc/en/Tutorial/Debounce
2. https://www.sparkfun.com/products/97
3. https://robu.in/product/dc-geared-motor-encoder-12-rpm-120n-cm-12v-spg30e-270k/
4. https://solarbotics.com/product/k_cmd/
5. https://www.sparkfun.com/products/9939
6. https://www.mathworks.com/matlabcentral/fileexchange/58257-unified-tuning-of-pid-load-frequency-controller-for-multi-area-power-systems-via-imc

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?
     Min: 6g, Typ: 7.2g
   - What is the sensor's dynamic range?
   - Min: -3gto +3g, Typ:- 3.6g to +3.6g
   - 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?
     The capacitor makes sure the accelerometer functions aren't affected by any noise or sudden variations from power supply. This is because capacitor doesn't allow a sudden change of voltage across its terminals by releasing charged energy when such a change occurs.
   - Write an equation for the sensor's transfer function.
     a=(V-Vo)/s
     s=sensitivity
     Vo=voltage at 0 acceleration
     V=measured voltage at acceleration a
   - What is the largest expected nonlinearity error in g?
     Min: 0.3*6g=0.180g
     Typ: 0.3*7.2g = 0.216g
   - How much noise do you expect in the X- and Y-axis sensor signals when the sensor is excited at 25 Hz?
     150 * sqrt(25) = 750g
   - How about at 0 Hz? If you can't get this from the datasheet, how would you determine it experimentally?
     0g. By measuring output when sensor is still.

2. Signal conditioning
   - Filtering
     - What problem(s) might you have in applying a moving average?
       A spiking noise in the data can push the moving average to an extreme value.
       If not spaced correctly, one could lose valuable data.
     - What problem(s) might you have in applying a median filter?
       Cannot capture peak features
       Computationally expensive than moving average
   - Opamps
     - 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.

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



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

Fig. 1 Opamp gain and offset circuit

(V2-V1)/Ri=(Vout-V2)/Rf
So, Vout=Rf/Ri*(V2-V1)+V2, ie, Vout=(1+Rf/Ri)* V2 - Rf/Ri * V1

Change in Vout=5V-0V=5V
In first case, change in Vin=1V-(-1.5)V=2.5V

Change in Vout/Change in Vin=1+Rf/Ri=5V/2.5V=2
Implies, Rf/Ri=1, and Vout=2*V2-V1

For the given range to be possible, V2 is the input voltage.
With end conditions, Vout=5, Vin=V2=1 and Vout=0, Vin-V2=-1.5, we get V1=-3, consistent. Hence, sensor can be calibrated.

In case of uncalibrated sensor, Change in Vin=2.5V-(-2.5V)=5V
Implies, Rs/Ri+1=1, and Rf/Ri=0
That is, either Ri is infinite or Rf is zero, in which case current across Rf would be infinite. Here, it cannot be calibrated

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.

Input to proportional term=current error=desired position-current position
Input to derivative term=change in error= current error-previous error
Input to integral term=total error=sum of errors from the past
To give digital input to each of these components, they are suppressed to an int value between 0 and 255. However, in the application of DC motor we only feed in a PID Term containing inputs and weights from all three terms.

- If the system you want to control is sluggish, which PID term(s) will you use and why?
  Proportional term decides the speed of control, it recommends higher change farther the desired value of parameter. If control is slow, so I'd increase Kp. However, if it is too high, system can overshoot.
- 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?
  Kd, the differential term manifests as differential error, as it decided what to do with differentials in error(current error-previous error). To decrease this steady state error, I'll increase Kd to increase sensitivity to 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?
  I'll use Ki, the integral term. Ki affects the overshoot, since it responds to errors summed over a past, ie, average offset after stabilisation without caring about small steady state variations.

Progress Review

We've installed required software on the workhorse computer at MRSD lab. We've zeroed in on three data sets(SEMAINE, IEMOCAP and Microsoft MISC) that fits the intent and conditions of our project, and we've decided to go with SEMAINE first owing to its diversity, large volume and single speaker. We've come across an end to end bimodal emotion recognition paper that we find suitable to get started on. Our aim is to implement it by the first Progress Review.

Codes

Code for Potentiometer Control & Debouncing:

```
int potpin = A0;   // select the input pin for the potentiometer
int inpin0 = 2;
int reading1=0;
int state_val=0;   // the current reading from the input pin
int previous1 = LOW;    // the previous reading from the input pin
long time = 0;         // the last time output pin was toggled
long debounce = 200;
int pwmpin=6;
int val=0;
int bri;
void setup() {
  //state1 set up
  analogReference(DEFAULT);
  pinMode(pwmpin,OUTPUT);
  Serial.begin(9600);
}

void loop()
{

  reading1 = digitalRead(inpin0);
  //Serial.println(reading1);
  if(reading1 != previous1)
  {
    if (reading1 == HIGH)
    {
      if(state_val==1)
        state_val=0;
      else
        state_val=1;
    }
    delay(50);
  }
  previous1=reading1;
  if (state_val==1)
  {
  potentiometer();
  }
```

```
  else
  {

  }
}

void potentiometer()
{
  //Serial.println("blah");
  val=analogRead(potpin);
  //Serial.println(val);
  bri=map(val , 0 , 1023 , 0, 255 );
  analogWrite(pwmpin,bri);
  Serial.println(bri);
}
```

Code for PID Control:

```
int PWM1 = 6;// this is the PWM pin for the motor for how much we move it to correct for its error
int InA1 = 7;//these pins are to control the direction of the motor (clockwise/counter-clockwise)
int InB1 = 3;

double setpoint = 100;//I am setting it to move through 100 degrees
double Kp = 0.32;// you can set these constants however you like depending on trial & error
double Ki = 0.1;
double Kd = 0.3;
double count = 0; //set the counts of the encoder
double currentangle = 0;//set the angles
boolean A,B;
byte state, statep;
float last_error = 0;
float error = 0;
float changeError = 0;
float totalError = 0;
float pidTerm = 0;
float pidTerm_scaled = 0;// if the total gain we get is not in the PWM range we scale it down so
that it's not bigger than |255|


void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT);//encoder pins
```

```
  pinMode(3, INPUT);
  attachInterrupt(digitalPinToInterrupt(encodPinA1),getFeedback,CHANGE);//interrupt pins for
encoder
  pinMode(PWM1, OUTPUT);
  pinMode(InA1, OUTPUT);
  pinMode(InB1, OUTPUT);

}

void loop(){

  PIDcalculation();// find PID value

  if (currentangle < setpoint) {
    digitalWrite(InA1, LOW);// Forward motion
    digitalWrite(InB1, HIGH);
  } else {
    digitalWrite(InA1, HIGH);//Reverse motion
    digitalWrite(InB1, LOW);
  }

  analogWrite(PWM1, pidTerm_scaled);
  delay(100);
}

void PIDcalculation(){
  currentangle = count;//count to angle conversion
  error = setpoint - currentangle;
  changeError = error - last_error; // derivative term
  totalError += error; //accumalate errors to find integral term
  pidTerm = (Kp * error) + (Ki * totalError) + (Kd * changeError);//total gain
  pidTerm = constrain(pidTerm, -255, 255);//constraining to appropriate value
  pidTerm_scaled = abs(pidTerm);//make sure it's a positive value
  last_error = error;
}

void getFeedback() //these functions are for finding the encoder counts
{
  A = digitalRead(encodPinA1);
  B = digitalRead(encodPinB1);

  if (((A==HIGH)&&(B==HIGH)) || ((A==LOW)&&(B==LOW))) state = 1;
  else state = 0;
```

```
  if (state){
    count++;
  }
  else count--;
}
```