Sensors and Motors Lab

Individual Lab Report 1

By **Rahul Ramakrishnan**

Team F:

Yuchi Wang

Pulkit Goyal

Pratibha Tripathi

Danendra Singh

13 October 2017

1. Individual Progress

My primary responsibilities for this lab we're to wire and control the servo by taking inputs from the Potentiometer and to setup the hardware, develop a code for the DC Motor Velocity and Position Control by taking inputs from the GUI during manual control (later part done in collaboration with Yuchi). I also worked on the troubleshooting part during integration with the other sensors and motors. The final system is shown in the fig.1 below.

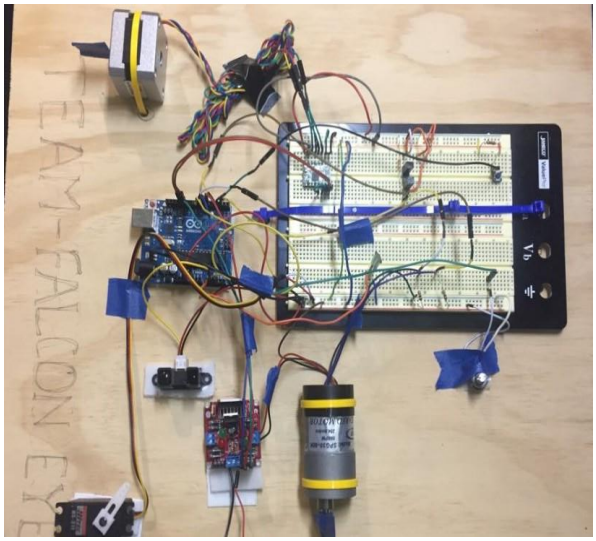

Fig.1 Final system

1.1. Potentiometer

The Potentiometer values were read from an Analog pin on the Arduino and an Average Filter was implemented on the incoming Potentiometer values to receive a reliable input. The potentiometer should be wired so that its two outer pins are connected to power (+5V) and ground, and its middle pin is connected to analog input 0 on the board.

Fig.2 Potentiometer
[Retrieved from https://en.wikipedia.org/wiki/Potentiometer]

For the Servo motor control, the filtered Potentiometer value was directly mapped to an angle between 0 – 180 degrees. For a change of Potentiometer value, a mapping to a set angle is done according to Table 1.

Potentiometer readings	Servo motor angle
0	0
1023	180

Table 1: Potentiometer mapping to servo angle

1.2. Servo

Servo motors have three wires: power, ground, and control. The power wire is connected to the 5V pin on the Arduino board. The ground wire is connected to a ground pin on the board. The Servo motor control pin was attached to PWM pin 9 and the servo.h header file in Arduino was used. The code for this system is attached in the appendix.

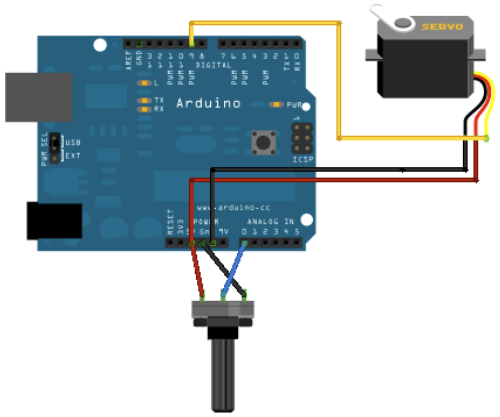

Fig.3 Circuit connection for servo and pot with Arduino
[Retrieved from https://www.arduino.cc/en/Tutorial/Knob]

1.3 DC Motor

The given DC Motor (Cytron SPG30-60K) is rated to operate at 12V with no load current less than 90mA and up to 1800mA when loaded [1]. Due to this possibility of high current, a Solarbotics DC Motor Driver was used to implement Arduino Logic commands in directing the Motor movements using I1 and I2 pins and controlling its speed through PWM from the Arduino using the enable pin [2]. External power supply was provided to the motor driver which is used to drive the motor.

1.4 Encoders

The motor has inbuilt encoders which was used to measure both the angle (i.e.) position and velocity. It is a two-channel encoder. The angle is measured as a direct function count where the count is incremented if both the channels of the encoders are in same state and decrements in count if the channels of the encoders are different states.

The encoder pins are attached to the interrupt pins and declared as interrupts with the mode as CHANGE. The value from the encoders are fed to the Arduino through pull-up resistors of 10k.

The velocity i.e. angular speed at any instant is calculated by the difference in the angles at previous and current instants of time divided by the time taken to reach the current angle from the previous position.

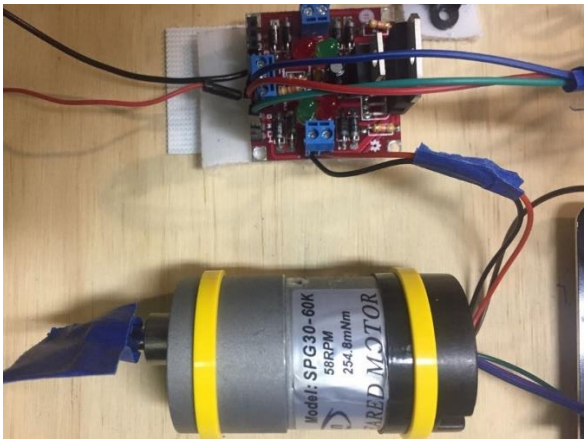

Fig.4 DC motor with driver

1.5 PID Control

The DC motor (Cytron SPG30-60K) was driven in two different modes. They were:

1. Velocity Control
2. Position Control

1.5.1 Velocity Control

For the velocity control, the velocity in RPM was set by the user in the GUI. The current speed was calculated by the change in angle/position at a fixed time

duration. The error was calculated by difference in the setpoint velocity and the current velocity.

$$error,\ e(t) = p(desired) - p(current)$$

This error is subtracted from previous error to calculate he change in error and is the error is added overtime to calculate the total error. All these parameters are used to calculate the correction term and is given to the PWM pin to control the speed.

The calibrated parameters for the velocity control were Kp =2, Ki =0.02, and Kd=0.1. The maximum overshoot was 3%.

1.5.2 Position Control

For the position control, the position in angles is set by the user in the GUI. The current position was calculated by the count from encoders. The error was calculated by difference in the setpoint angle and the current angle. This error is subtracted from previous error to calculate he change in error and is the error is added overtime to calculate the total error. All these parameters are used to calculate the correction term and is given to the PWM pin to control the position.

The calibrated parameters for the position control were Kp =5, Ki =0.00, and Kd=2. The maximum overshoot was 1 degrees.

The process of PID control was the following:

1. Get the position/velocity from user via GUI.
2. Measure current value (velocity or position) from Encoder Counts and Time.
3. [PWM] = PID (Setpoint, current value).
4. Send PWM and direction command to motor.
5. Goto 1.

2.0 Challenges

2.1 DC Motor PID control

One of the major challenges for me was the tune up process of the parameters for both position and velocity. It was tedious and iterative process and the most difficult part being that the calibrated parameters weren't working all the time.

There was so much uncertainty associated with the system due to vibrations, voltage drops etc.

Accounting for all these was an extremely tough task. Finally, we reached on a set of parameters which was an average of all the working parameters that we calibrated on before and it worked just fine. This was the case for both position and velocity.

2.2 Servo Motor

The other difficulty was to figure out why the DC motor wasn't working when plugged into PWM pin 10 while integrating the system as it was working fine as a sub-system. The problem was that when we used servo.h header, it allots both pin 9 and 10 for servos by default. This took me a while to debug but eventually had it working by switching it to other PWM pin.

3. Teamwork

Yuchi Wang: Force sensor, PID control
Pulkit Goyal: Graphical User Interface
Pratibha: Servo with IR
Danendra Singh: Stepper and Slot sensor

The entire team of 5 started working together for this lab assignment at our workbench which made possible the resolving of queries and conflicts at one place instantaneously which wouldn't have been possible if we had worked independently. Yuchi was in-charge of software integration combining codes from all the members. Pulkit had been working on the entire GUI part and made it ready for interfacing with the Arduino. Pratibha and Danendra did the role of combining all the sensors and motors on one board to be ready for integrated testing. I took the role of trouble shooting errors while integration of hardware and software. I also involved myself in the software tuning of the control loops with Yuchi.

4. Future Plans

We just got the husky batteries and communication cable delivered. We are currently working on the husky to get it to move. We are having a lot of driver

issues and are trying to resolve them. Meanwhile we are planning to have a few test flights with the new drones we got from our sponsor (Parrot Bebop 2). We have to figure out the support for the drone's SDK. We are waiting for acquiring a Velodyne puck to get started on it. Studies are being done on the various algorithms used for SLAM and path planning. We do have a backup option of getting the husky from NREC if this husky fails to work. Our next milestone is to make the husky move in the given direction for a given distance.

References

[1] "DC Geared Motor with Encoder SPG30E-60K," Cytron Technologies, [Online]. Available: http://www.cytron.com.my/p-spg30e-60k

[2] "Solarbotics L298 Compact Motor Driver Kit," Solarbotics, [Online]. Available: https://solarbotics.com/product/k_cmd/ .

[3] https://www.arduino.cc/en/Tutorial/Knob

[4] https://hackaday.io/project/11382-dc-motor-speed-control-with-pid

Task 7 (Sensors and Motor Control Lab) Quiz

1. **Reading a datasheet**

o What is the sensor's range?

· +-3g (minimum) and +/- 3.6g (typical)

o What is the sensor's dynamic range?

20log(0.707*3.6/150ug) (u-micro)

o 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?

· Capacitor is used to decouple the accelerometer from noise on the power supply. A single 0.1 uF capacitor, CDC, placed close to the ADXL335 supply pins is used to achieve this.

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

· 1.5 + 0.3x

o What is the largest expected nonlinearity error in g?

· +/-0.3% of full scale range

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

· 750ug/rms

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

- Set it to 0Hz
- Measure deviation from nominal voltage
- Convert the deviation into RMS

2. **Signal conditioning**

o Filtering

a. What problem(s) might you have in applying a moving average?

- Ignores complex relationships in data [different frequencies].
- Slow response time for large windows.

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

- High computational cost.
- May not be suitable for real-time applications where you want the process to be fast [Incase if large window size].

o Opamps

1. 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.

V2 = Vin, V1 = Vref
Since the range of uncalibrated sensor is -1.5 to 1.0V,
Assuming,
Ri = 1
Rf = 1
Calculating using this gives:
Vref = -3 V.

Since the range of uncalibrated sensor is -2.5 to 2.5V,
Solving the system of equations with 2 equations and 3 variables returns no solution.

[Calculations done using Wolfram Alpha].

3. Control

o 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.

- Proportional term: Difference between desired and current position (error)
- Derivative term: Approximately equal to difference between previous successive error values (change in error)
- Integral term: sum of previous errors (total error)

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

I will use a P term as output is proportional to error and large gain is achievable.

o 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 will use the integral term as it is proportional to sum of errors and will give that extra magnitude to reach the desired output.

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

Derivative doesn't allow rapid changes in output and provides damping. It prevents drastic changes in output rate and hence prevents overshoot (counteracting term).

APPENDIX

1) Servo with Potentiometer

```
void loop()
{
  val = analogRead(potpin);           // reads the value of the potentiometer (value between 0 and
1023)
   Serial.println(val);

  double valr = map(val, 0, 1023, 0, 255); // scale it for the servo (value between 0 and 180)
  Serial.print("servo degree: ");
if (auto_read == 2)
{
  servo.write(double(val_read)); // sets the servo position according to the scaled value
}
else
{
  servo.write(double(valr));
}
}
```

2) DC motor PID_position and PID_velocity

```
volatile double setpoint ;
 double tar_spd ;
double Kpd = 5;// you can set these constants however you like depending on trial & error
double Kid= 0;
double Kdd = 2;
double Kps = 2;// you can set these constants however you like depending on trial & error
double Kid= 0.02;
double Kdd = 0.1;
double distance = 0;
float last_error = 0;
float error = 0;
float changeError = 0;
float totalError = 0;
float pidTerm = 0;
float pidTerm_scaled = 0;  // PWM range we scale it down so that it's not bigger than 255
```

```
void loop()
{
if(guiinput == 1)
{
void loop_PID_SPD(tar_spd);
}
else
{
void loop_PID_POS(setpoint);
}
}

void loop_PID_SPD()
{
  PIDcalculation_SPD();  // find PID value
   if (tar_spd < 0)
 {
    digitalWrite(dir1, LOW);  // Forward motion
    digitalWrite(dir2, HIGH);
    delay(50);
  }
  else
{
    digitalWrite(dir1, HIGH);  // Forward motion
    digitalWrite(dir2, LOW);
    delay(5);
  }
 analogWrite(pwm, pidTerm_scaled);
  delay(100);
}

void loop_PID_POS()
{
  PIDcalculation_POS();   // find PID value

  if (angle > setpoint) {
    digitalWrite(dir1, LOW);    // Forward motion
    digitalWrite(dir2, HIGH);
  }
  else
{
    digitalWrite(dir1, HIGH);   // Forward motion
    digitalWrite(dir2, LOW);
  }
```

```
  analogWrite(pwm, pidTerm_scaled);
  delay(100);
}

void PIDcalculation_POS()
{
  angle = (1 * count);   //count to angle conversion
  error = setpoint - angle;

  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 loop_step()
{
vals = digitalRead(ruptPin);
    if (vals==0)
    {

    a++;
    digitalWrite(stp, HIGH);
    delay(5);
      digitalWrite(stp, LOW);
     delay(5);
    }
}


void PIDcalculation_SPD()
{
  prev_angle = (1 * count);  //count to angle conversion
  delay(50);
  angle = (1 * count);

  curr_spd = (angle - prev_angle)/ 50;
  error = abs(tar_spd) - abs(curr_spd);

  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 Achange()   //these functions are for finding the encoder counts
{
  A = digitalRead(2);
  B = digitalRead(8);

  if(A==B)
  {
    count++;
  }
  else
  {
    count--;

  }
}
void modechange()
{

  currtime = millis();
flag = true;
}
```