Individual Lab Report 1

Name : **Ardya Dipta Nandaviri Giri**
Team B : **Space Jockey**
Team members : **Brian Boyle, Nathaniel Chapman, Songjie Zong**
ILR01
Submission date : **October 10,13**

1. Introduction

This report gives a brief description of our team progress along with my individual contribution so far towards the MRSD project and Motor Control lab assignment (Task 6). The objectives in this lab are: reading the sensors data and integrates it to motors such as servo, DC and stepper motor. Figure 1 is the full view of our team's circuit, which consists: (1) Arduino UNO as a main controller board; (2) Infra Red Sharp GPD12 Range Finder,(3) potentiometer that is connected with an arm made in acrylic, (4) force sensor as sensors; and (5) DC motor, (6) servo motor, and (7) stepper motor for the actuators.

2. Individual Progress

2.1. Speed Control Programming for the DC motor

I applied my knowledge I learned from "Manipulation Control and Mobility" course about PID. To reach the speed desired, I measured the error from the current speed by reading encoders. The speed of the motor is measured in RPM (rotation per minute) unit. From the experiment, the DC motor's revolution takes 360 pulses of encoder. Hence, to measure the speed I counted pulses generated every 100ms, divided it by 360 and multiplied it by 60000 to get RPM unit. After getting the current speed value, I compared the desired value and then calculate the errors. This error value will be an input for the PID equation to correct the errors.

2.2. Position Control Programming for the DC Motor

Working on position control for DC motor is similar to speed control. This time I used the encoders to sense the DC motor position value. Since it generates 360 pulses of encoder for each revolution, we knew that each pulse represents 1 degree of movement. By counting the encoder pulses, we were able to measure the error from the desired position and then calculate the error using the same PID method in section 2.1.

2.3. Programming the range finder GPD12

The range finder sensor is a variable resistor with its value depends on the feedback of infrared transmission. From the experiment made by luckylarry.co.uk , the distance can be obtained by calculating the sensor's voltage using "pow" command in Arduino, that lets us raise a specified number by a power/ exponent. The exponent chart is shown in figure 2.

d. Programming the servo motor using Arduino library.

Arduino has sets of libraries; servo library is one of them. By using this library, we can control the servo position by giving input position.

e. Things I have learned

It was a great learning experience working on this lab. In this project I learned how to control the speed and position of DC motor using PID. This PID control is important for our future project. I also learned how to measure the distance using infrared sensor GPD12 and how to reduce the noise of the sensors.

3. Challenges / issues

3.1. Fine tuning the PID control

It is not an easy task to find the most stable tuning for the PID control. If the control is unstable, the DC motor will have overshooting and oscillation. It was really hard for the first time to fine the correct value of Kp KI and Kd. After did a few testing, I found a good value for the Kp Kd and Ki. For the speed control, I got accuracy of +-5RPM. While for the position control, I got accuracy of 5 degrees.

3.2. Motor jammed

While working on this lab, somehow the DC motor jammed so that it couldn't move and made the L297 chip hot. At the first time, I thought that the motor controller is broken. But later on, I found that the DC motor is the one that actually broke since it wouldn't move when I connected it directly to DC power supply without using motor controller. It seems that the gearbox inside jammed so that it could not move. After replaced the motor, I could continue on working.

3.3. Sensor noise

There was noise in sensor reading. The GUI showed that the sensor value was bouncing. In order to get the correct value, our team used the "Smoothing" method from Arduino web site. This method reads repeatedly from an analog input, calculating a running average and printing it.

4. Cross-referencing with the work of fellow team members

The work for this lab was evenly divided in such a manner that each person would have enough experience in both electronics and software programming. Songjie helped me to develop the PID control for position control on DC motor. We both worked on tuning the PID correction value. Nate integrated my code to the main program. Brian worked on the GUI and also helped Nate developing the serial connection of the main program to his GUI.

5. Plans / goals for following week

My future work for the project is to design and develop the electrical system of the robot including the power, main processor, motor controllers and sensors. For the next following week I will help the team to do trade studies in battery and motor, and also start working in power distribution board. Designing the power distribution board can be challenging since we still do not know precisely how much power will we need. The power of the motors needed is also yet to be determined.

6. Figure(s)

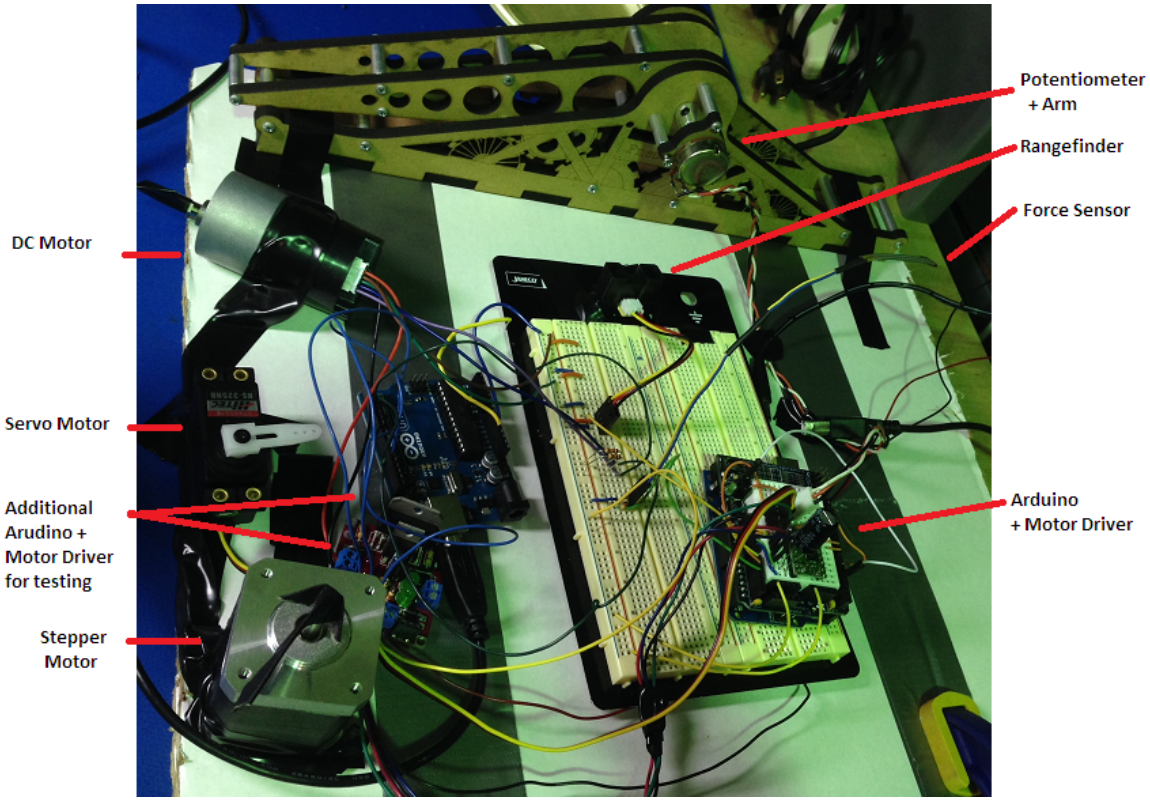

Figure 1. Circuit board for task 6

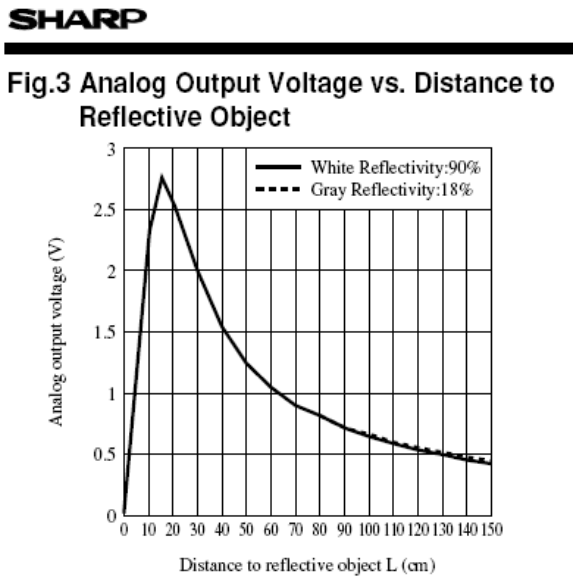

Figure 2. GPD12 Output from datasheet

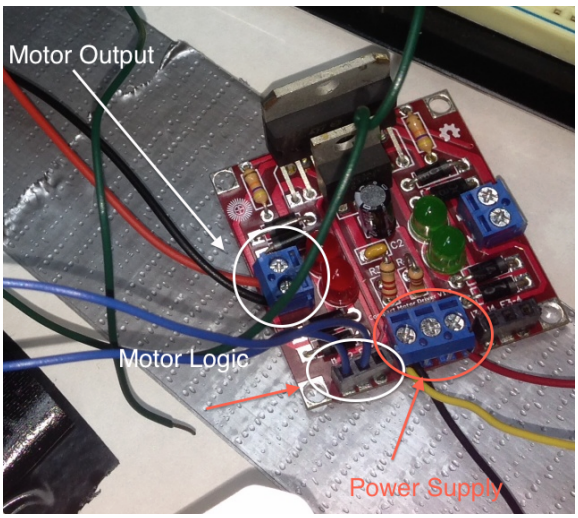

Figure 3. Motor Controller

6. Arduino programming

6.1. Speed Control

```
// Pin assignment for motor
#define encoder0PinA 2 // encoder channel A
#define encoder0PinB 3 // encoder channel B
#define En1  10    //enable pin Motor DC, connected to PWM controlto control speed
#define L1   11   // Logic input 1 for motr DC
#define L2   12 // Logic input 2 for motor DC
#define LOOPTIME       100                 // PID loop time every 100ms
unsigned long lastMilli = 0;               // loop timing
unsigned long lastMilliPrint = 0;            // loop timing
int speed_req = 0;                      // speed (Set Point)
int motorSpeedCurr = 0;                       // speed (actual value)
int PWM_val = 0;                        // (25% = 64; 50% = 127; 75% = 191; 100% = 255)
volatile long count = 0;                  // rev counter
float Kp =   .4;                       // PID proportional control Gain
float Kd =    1;                       // PID Derivitave control gain
long encoder0Pos = 0;
int degree = 0 ;                         // degree of DC motor shaft
int revolution = 0 ;                     // how many times the revolutions
float pidTerm = 0;                        // PID correction
int error=0;                           // error calculation = target value- actual value
static int last_error=0;                  // Last error
static long last_count = 0;                  // last count


////////////////////////////////////////// SETUP
PINS//////////////////////////////////////////
void setup()
{
// Setup for the the motor controller
  pinMode(En1, OUTPUT);
  pinMode(L1, OUTPUT);
  pinMode(L2, OUTPUT);
  pinMode(encoder0PinA, INPUT);
  pinMode(encoder0PinB, INPUT);

// encoder pin on interrupt 0 (pin 2)
  attachInterrupt(0, doEncoderA, CHANGE);

// encoder pin on interrupt 1 (pin 3)
  attachInterrupt(1, doEncoderB, CHANGE);
// serial
```

```
  Serial.begin (9600);

  // initial of PWM
  analogWrite(En1, PWM_val);

}


// function to read encoder in interrupt
void doEncoderA(){

  // look for a low-to-high on channel A
  if (digitalRead(encoder0PinA) == HIGH) {
    // check channel B to see which way encoder is turning
    if (digitalRead(encoder0PinB) == LOW) {
      encoder0Pos = encoder0Pos + 1;         // CW
    }
    else {
      encoder0Pos = encoder0Pos - 1;         // CCW
    }
  }
  else   // must be a high-to-low edge on channel A
  {
    // check channel B to see which way encoder is turning
    if (digitalRead(encoder0PinB) == HIGH) {
      encoder0Pos = encoder0Pos + 1;          // CW
    }
    else {
      encoder0Pos = encoder0Pos - 1;          // CCW
    }
  }

}

void doEncoderB(){

  // look for a low-to-high on channel B
  if (digitalRead(encoder0PinB) == HIGH) {
   // check channel A to see which way encoder is turning
    if (digitalRead(encoder0PinA) == HIGH) {
      encoder0Pos = encoder0Pos + 1;         // CW
    }
    else {
      encoder0Pos = encoder0Pos - 1;         // CCW
    }
  }
```

```
  // Look for a high-to-low on channel B
  else {
    // check channel B to see which way encoder is turning
    if (digitalRead(encoder0PinA) == LOW) {
      encoder0Pos = encoder0Pos + 1;         // CW
    }
    else {
      encoder0Pos = encoder0Pos - 1;         // CCW
    }
  }
}


//
void motor_dc(char c)
{
  if(c=='f') // forward
  {
    digitalWrite(L1, LOW);
    digitalWrite(L2, HIGH);
  }
  else if(c=='b') //backward
  {
    digitalWrite(L1, HIGH);
    digitalWrite(L2, LOW);
  }
}


void getMotorData()  {                                  // calculate speed

  motorSpeedCurr = ((encoder0Pos - last_count)*(60*(1000/LOOPTIME)))/(360);         //
360 pulse each revolution
  last_count = encoder0Pos;
}

int updatePid(int command, int targetValue, int currentValue)   {           // compute PWM
value

  error = abs(targetValue) - abs(currentValue);
  pidTerm = (Kp * error) + (Kd * (error - last_error));
  last_error = error;
  return constrain(command + int(pidTerm), 0, 255);

}
```

```
// Motor Speed controller function
void motor_speed(int speed_desired)
{
 // if the speed required is negative, then the motor rotate backwards
 if (speed_desired < 0)
 {
  motor_dc('b');
 }
 else {motor_dc('f');}

// getting the speed data every LOOPTIME
 if((millis()-lastMilli) >= LOOPTIME)   {                    // enter timed loop
  lastMilli = millis();
  getMotorData();                                            // calculate speed
  PWM_val= updatePid(PWM_val, speed_desired, motorSpeedCurr);                    //
compute PWM value
  analogWrite(En1, PWM_val);                                 // send PWM to motor
 }

 Serial.print("Speed RPM: ");
 Serial.println(motorSpeedCurr);

}

////////////////////////////////
void loop()
{
 motor_speed(100);  // motor speed controller, desired input in RPM units (Max 80)
}
```

6.2. Position Control

```
// Pin assignment for motor
#define ENCODER_A 2 // encoder channel A
#define ENCODER_B 3 // encoder channel B
#define EN1  10    //enable pin Motor DC, connected to PWM controlto control
speed
#define L11   11   // Logic input 1 for motr DC
#define L12   12 // Logic input 2 for motor DC

#define FORWARD  1
#define BACKWARD 0

int PWM_val = 0;                    // (25% = 64; 50% = 127; 75% = 191; 100% =
255)
float Kp =   .2;                   // PID proportional control Gain
float Kd =    1;                   // PID Derivitave control gain
long encoderPosition = 0;
static int last_error=0;             // Last error
static long last_count = 0;             // last count

// variable for motor position
int degree_req = 0;                 // speed (Set Point)
int motorDegCurr = 0;                   // speed (actual value)



////////////////////////////////////////// SETUP
PINS//////////////////////////////////////////
void setup()
{
// Setup for the the motor controller
  pinMode(EN1, OUTPUT);
  pinMode(L11, OUTPUT);
  pinMode(L12, OUTPUT);
  pinMode(ENCODER_A, INPUT);
  pinMode(ENCODER_B, INPUT);

// encoder pin on interrupt 0 (pin 2)
  attachInterrupt(0, doEncoderA, CHANGE);

// encoder pin on interrupt 1 (pin 3)
  attachInterrupt(1, doEncoderB, CHANGE);
// serial
  Serial.begin (9600);
```

```
  // initial of PWM
  analogWrite(EN1, PWM_val);

}
////////////////////////////////////////// SETUP PINS
END//////////////////////////////////////////


// function to read encoder in interrupt
void doEncoderA(){

  // look for a low-to-high on channel A
  if (digitalRead(ENCODER_A) == HIGH) {
    // check channel B to see which way encoder is turning
    if (digitalRead(ENCODER_B) == LOW) {
      encoderPosition = encoderPosition + 1;         // CW
    }
    else {
      encoderPosition = encoderPosition - 1;         // CCW
    }
  }
  else   // must be a high-to-low edge on channel A
  {
    // check channel B to see which way encoder is turning
    if (digitalRead(ENCODER_B) == HIGH) {
      encoderPosition = encoderPosition + 1;         // CW
    }
    else {
      encoderPosition = encoderPosition - 1;         // CCW
    }
  }

}

void doEncoderB(){

  // look for a low-to-high on channel B
  if (digitalRead(ENCODER_B) == HIGH) {
   // check channel A to see which way encoder is turning
    if (digitalRead(ENCODER_A) == HIGH) {
      encoderPosition = encoderPosition + 1;         // CW
    }
    else {
      encoderPosition = encoderPosition - 1;         // CCW
    }
```

```
  }
  // Look for a high-to-low on channel B
  else {
   // check channel B to see which way encoder is turning
   if (digitalRead(ENCODER_A) == LOW) {
    encoderPosition = encoderPosition + 1;         // CW
   }
   else {
    encoderPosition = encoderPosition - 1;         // CCW
   }
  }
}


void setMotorDir(uint8_t dir){
 if(dir == FORWARD) {
  digitalWrite(L11, LOW);
  digitalWrite(L12, HIGH);
 }else {
  digitalWrite(L11, HIGH);
  digitalWrite(L12, LOW);
 }
}


void calcMotorData() {                                    // calculate speed
  motorDegCurr = (encoderPosition) ;
  //motorSpeedCurr = ((encoderPosition -
last_count)*(60*(1000/LOOPTIME)))/(360);        // 360 pulse each revolution
  last_count = encoderPosition;
}

int updatePid_position(int command, int targetValue, int currentValue)   {          //
compute PWM value

 int error = abs(targetValue) - abs(currentValue);
 if (error <0)
 {
  setMotorDir(BACKWARD);
  error = error * -1;
 }
 else
 {
  setMotorDir(FORWARD);
 }
```

```
 float pidTerm = (Kp * error) + (Kd * (error - last_error));
 last_error = error;
 return constrain(command + int(pidTerm), 0, 255);

}

void motor_position(int degree_desired)
{

   calcMotorData();                                          // calculate speed
   PWM_val = updatePid_position(PWM_val, degree_desired, motorDegCurr);
// compute PWM value
   analogWrite(EN1, PWM_val);                                  // send PWM to motor

}

////////////////////////////////
void loop(){
 motor_position(120);
}
```

6.3. Servo, DC Motor and GPD12 Sensor Programming

```
#include <Servo.h>
Servo myservo;

// Pin assignments
int IRpin = A0;                    // analog pin for reading the IR sensor
int En1 = 8; //Enable pin for motor 1
int L1 = 9; // Logic 1 for motor 1
int L2 = 10; // Logic 2 for motor 2
int servopin = 2; // Servo pin


// Global variable


void setup() {
 Serial.begin(9600);                // start the serial port
 pinMode(En1, OUTPUT);
 pinMode(L1, OUTPUT);
 pinMode(L2, OUTPUT);
 myservo.attach(servopin);
}


void motor_dc(char c)
{
 if(c=='f') // forward
 {
  digitalWrite(En1, HIGH);
  digitalWrite(L1, LOW);
  digitalWrite(L2, HIGH);
 }
 else if(c=='b') //backward
 {
  digitalWrite(En1, HIGH);
  digitalWrite(L1, HIGH);
  digitalWrite(L2, LOW);
 }
}

void loop()
{
```

```
 //============================================== Using GPD
12============================================================
=======
//  float volts = analogRead(IRpin)*0.0048828125;   // value from sensor *
(5/1024) - if running 3.3.volts then change 5 to 3.3
//  float distance = 65*pow(volts, -1.10);         // worked out from graph 65 =
theretical distance / (1/Volts)S - luckylarry.co.uk
//  Serial.println(distance);                    // print the distance
//  delay(100);                             // arbitary wait time.

//============================================================
============================================================
===


//==============================================Usinng motor
controller==================================================
===
//  motor_dc('f');
//  delay(2000);
//  motor_dc('b');
//  delay(2000);
//============================================================
============================================================
===

// ==============================================Usinng Servo
motor=======================================================
//myservo.write(90);
//delay(1000);
//myservo.write(0);
//delay(1000);
//============================================================
============================================================
===

}
```