Sensors and Motor Control Lab

ILR - 1

Amit Agarwal

Team A

Amit Agarwal, Harry Golash, Yihao Qian, Menghan Zhang, Zihao (Theo) Zhang

October 14, 2016

Table of Contents

1. Individual Progress

For the Sensors and Motor Control Lab, I was responsible for 3 main tasks. The tasks were interfacing the IR sensor with the DC Motor, implementing and tuning the PID controls for position and velocity control, and the hardware and software integration of all other sensors and motors.

1.1 DC Motor Control – Setup

To set up the DC motor to work with the IR sensor (SHARP sensor), the first task that I undertook was to assemble the DC motor driver. The DC motor driver used was a Compact L298 Motor Driver by Solarbotics. A kit was provided, which contained a printed circuit board L298 motor chip driver, a LM2937 voltage regulator chip, a 22μF 50V capacitor, a 0.1μF capacitor, a 3-position terminal block, 2 2-position terminal blocks, 2 47k enable pull-up resistors, 2 red LEDs, 2 green LEDs, 8 1N4007 EMF-protection diodes, 2 3-position interface socket strips. All the components were soldered to the PCB according to the instructions provided with the kit.

After the DC motor driver was assembled, the SHARP 2Y0A02 IR sensor was set up. The IR sensor has 3 wires coming out – the first one was used for ground, the second one was used for 5 V (V_{cc}) from the microcontroller, and the third one was used for reading analog signals through pin A3 on the microcontroller. The DC motor used was a Cytron Technologies SPG30-60K rated for 58 RPM and 254.8 mNm. This particular motor has an inbuilt encoder. There are 6 wires that need to be connected to the DC motor – wire 1 was connected to the negative output from the DC motor driver, wire 2 was connected to the positive output from the DC motor driver, wire 3 was connected to the 5 V (V_{cc}) through the microcontroller for voltage reference, wire 4 was connected to ground through the microcontroller, wire 5 was connected to the pin 2 of the microcontroller for channel A of the encoder, and wire 6 was connected to the pin 3 of the microcontroller for channel B of the encoder. A 12 V input was provided to the DC motor driver through the DC power supply. A 5 V reference voltage was also provided to the DC motor driver through the microcontroller. Pins L1, L2 and E1-2 of the DC motor driver were connected to pins 7, 8 and 11 of the microcontroller to set the direction of the DC motor and the to provide it with an appropriate PWM output. The microcontroller used was an Arduino Uno.

1.2 DC Motor Control – Position

Basic PID code was implemented through the Arduino IDE in a file with a .ino extension. The encoder value was read through the program by using a library named Encoder.h. To convert the value to degrees, the value was modulus with 700 and then was divided by 700 and multiplied by 360. The code was structured in a manner such that the IR sensor would read a distance value converted to degrees, set it as the setpoint for PID control, then provide PID optimized output to the motor until the setpoint had been achieved within an error of 5^o before reading another value from the IR sensor to repeat the process. The PID controller was provided with input from the encoder to calculate the error. To limit a continuous overshoot and cycling of the motor, if the degree input from the IR sensor was over 180^o, then it was subtracted from 360^o to provide the setpoint. After multiple iterations of tuning, the proportional control was set to 0.8, the integral control was set to 0.2 and the derivative control was set to 0.9. These values worked well for position control of the DC motor.

1.3 DC Motor Control – Velocity

Similar PID code was implemented for velocity control. A major portion of the code from position control was reused for velocity control. Similar conversion factors were used for the encoder and the IR sensor. The angular velocity was calculated in degrees per second using the millis() function to track time and two variables to track the initial and final positions of the encoder in each iteration of the PID loop. For velocity control, a value was read from the IR sensor to provide the setpoint velocity in degrees per second and then 100 iterations of PID control were run regardless of if the setpoint velocity was achieved or not before reading another value from the IR sensor. To compensate for the deadband of the DC motor at low PWM values, the motor was run at maximum PWM for 2 seconds to avoid the motor getting stalled. Some if conditions were set to appropriately handle edge cases of velocity and PID outputs.

1.4 Hardware and Software Integration

Firstly, all the individual circuits and code were gathered from all teammates. The circuit was assembled by assigning the pins on the Arduino to avoid any confusion with regarding the pin

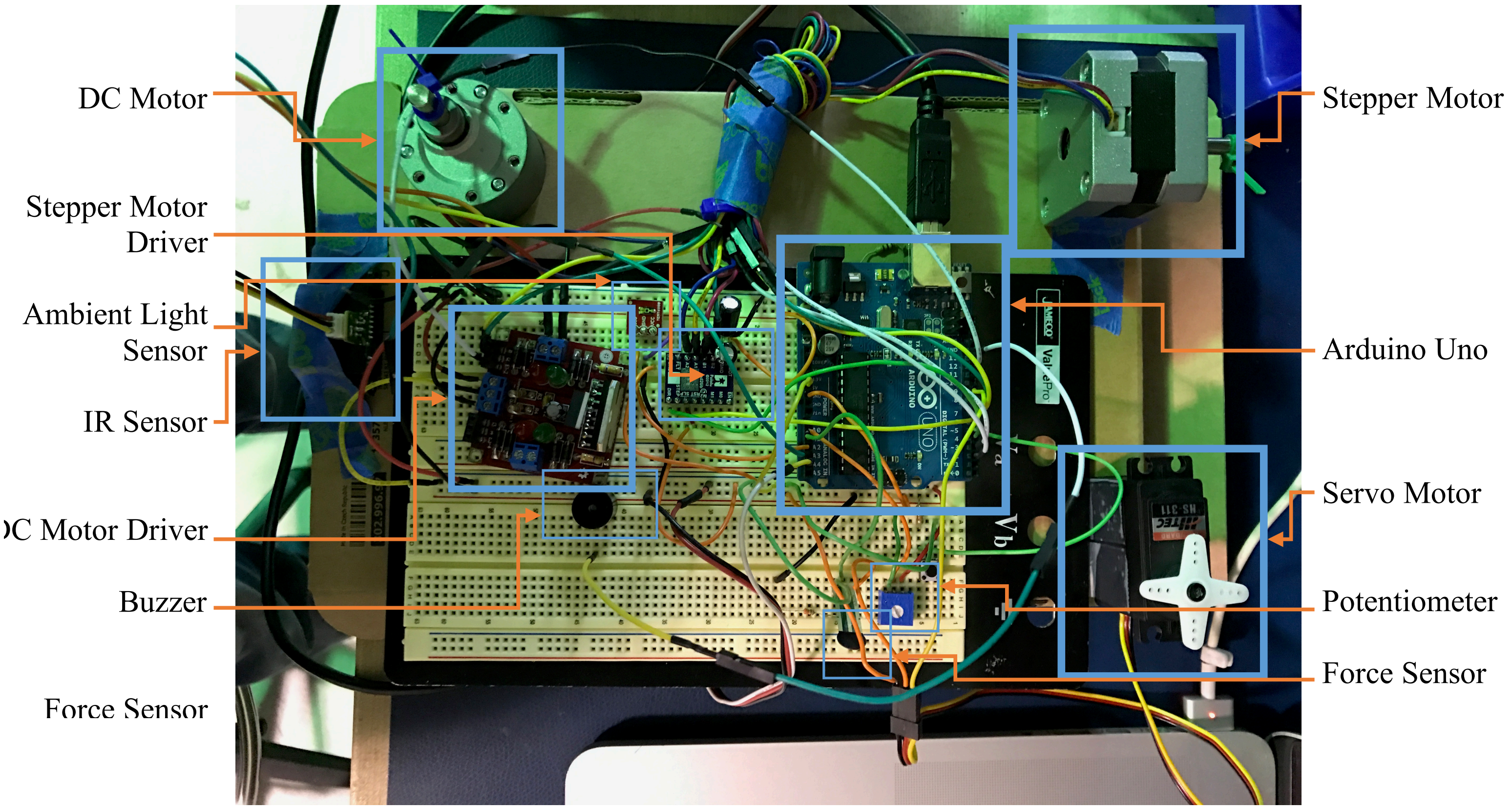

Figure 1. Hardware Layout

and component connections. After assigning pins, the wires were connected through a single breadboard to the components and the microcontroller as shown in figure 1. Then, each members' parts integrated one by one into a unified file. The variables and helper functions were carried over directly while avoiding duplicate names. The lines of code from the setup() function were also copied over to the unified setup() function. The lines of code from the loop() function were carried over into a separate function for each members' tasks. These functions were called through a switch case in the loop() function which would check the system state variable to decide which task is to be performed. Once the circuit and the code were in order, all components were fixed in used cardboard box with slots for sensors.

2. Challenges

There were no major challenges in this lab. A few minor problems are mentioned below.

2.1 Encoder Inconsistency

The encoder attached with the DC motor had inconsistent measurements. This could be caused by hysteresis or the unintended physical movement of the parts in the DC motor and encoder due to the vibration generated by the DC motor. To minimize this inconsistency, the conversion of the encoder values was recalibrated before the demo.

2.2 Stepper Motor

Due to a mistake on my teammate's part while selecting the capacitor for the stepper motor, when I was doing the integration, the stepper motor suddenly stopped working. This was solved by getting a replacement stepper motor by a TA.

3. Teamwork

The work on this lab was performed by all 5 team members. I was responsible for the DC motor position and velocity control with the IR sensor, the integration of the final system, and for presenting the demo. Harry was responsible for creating a car reverse sensor using an ultrasound sensor and applying the GUI to the integrated code provided by me. Yihao was responsible for the position control of the servo motor using the force sensor. Zihao was responsible for the position control of the stepper motor using the ambient light sensor. Menghan was responsible for the positon control of the servo motor using the potentiometer. All members worked well and in a timely manner.

4. Future Plans

The goals for the work to be accomplished by the team by the next lab demo is designing a mounting method and successfully receiving data from the 2 cameras and the radar sensor. I will be working with Harry in designing and prototyping the mounting method and the mounting rack.

5. References

[1] "Solarbotics L298 Compact Motor Driver Kit :: Solarbotics". Solarbotics.com. N.p., 2016. Web. 15 Oct. 2016.

Code

```
#include <Encoder.h>

#include <Servo.h>
#include <Stepper.h>
#define step_pin 6 // Definepin 3 as the steps pin
#define dir_pin 4 // Define pin 4 as the direction pin
#define light_pin A2 // Define A2 as the light sensor

Servo myservo;


// create servo object to control a servo
const int numReadings = 10;
int potpin = A0;  // analog pin used to connect the
potentiometer
int val;    // variable to read the value from the analog pin
int readings[numReadings];      // the readings from the analog
input
int readIndex = 0;              // the index of the current
reading
int total = 0;                  // the running total
int average = 0;

int potpinf = 1;
int valf;
int i;
const int numReadingsf = 10;
int readingsf[numReadingsf];      // the readings from the
analog input
int readIndexf = 0;              // the index of the current
reading
int totalf = 0;                  // the running total
int averagef = 0;

//Stepper
const int stepsPerRevolution = 200;
int lastStep = 0;
int currStep = 0;
Stepper myStepper(stepsPerRevolution, step_pin, dir_pin);

//Position PID
int enable_pin = 11;
int pin_l1 = 7;  //pin l1
int pin_l2 = 8;  //pin l2
int pin_e1 = 2;
int pin_e2 = 3;
const int irSense = A3;          // Connect sensor to analog pin
A3
int distance = 0;
int sample = 0;
double temp = 0;
double error = 0;
double error_last = 0, error_d = 0, error_i = 0;
//PID vars
double pid_in = 0;    //encoder value
double pid_out = 0;   //PWM from 0-255
double pid_set = 0;   //encoder value from IR through mapping
double pid_p = 0.8;
double pid_i = 0.2;
double pid_d = 0.9;
double new_pos_val;
//Encoder vars
Encoder enc_pos(pin_e1, pin_e2);
long pos_val  = -999;
int loop_check = 0;

//Velocity PID
unsigned int encoder0Pos = 0;
unsigned int tmp_Pos = 1;
double vtemp;
double vpid_set = 0;
double ang_vel;
long time_int = 0;
long init_pos = 0;
double vpid_in = 0, vpid_out = 0;
unsigned int vcount = 0;
double verror = 0;
double verror_last = 0, verror_d = 0, verror_i = 0;
double vpid_p = 4.5;
double vpid_i = 0.8;
double vpid_d = 1.5;


//debouncing
int buttonPin = 5;    // the number of the pushbutton pin


// Variables will change:

int buttonState;             // the current reading from the
input pin
```

```
int lastButtonState = LOW;   // the previous reading from the
input pin

// the following variables are unsigned long's because the time,
measured in miliseconds,
// will quickly become a bigger number than can be stored in an
int.
unsigned long lastDebounceTime = 0;  // the last time the output
pin was toggled
unsigned long debounceDelay = 50;    // the debounce time;
increase if the output flickers
int sysstate = 0;



void setup() {

  Serial.begin(9600);

  //Stepper
  pinMode(dir_pin, OUTPUT);
  pinMode(step_pin, OUTPUT);
  pinMode(light_pin, INPUT);
  myStepper.setSpeed(200);


  //pinMode(buttonPin, INPUT);
  myservo.attach(9);  // attaches the servo on pin 9 to the
servo object
  pinMode(potpinf, INPUT);
  pinMode(buttonPin, INPUT);
  myservo.attach(9);

  for (int thisReadingf = 0; thisReadingf < numReadingsf;
thisReadingf++) {
    readingsf[thisReadingf] = 0;
  }

  for (int thisReading = 0; thisReading < numReadings;
thisReading++) {
    readings[thisReading] = 0;
  }
  ang_vel = 0;
  init_pos = 0;
}
```

```
void portential()
{

  total = total - readings[readIndex];
  val = analogRead(potpin);            // reads the value of the
potentiometer (value between 0 and 1023)

  readings[readIndex] = map(val, 0, 1023, 500, 2400);     //
scale it to use it with the servo (value between 0 and 180)
  // add the reading to the total:
  total = total + readings[readIndex];
  // advance to the next position in the array:
  readIndex = readIndex + 1;
  if (readIndex >= numReadings) {
    // ...wrap around to the beginning:
    readIndex = 0;
    Serial.print("pot");
    Serial.println(average);
    myservo.writeMicroseconds(average);
  }

  // calculate the average:
  average = total / numReadings;
  // send it to the computer as ASCII digits


  //Serial.println(val);

  //myservo.write(val);                  // sets the servo
position according to the scaled value
  delay(10);                           // waits for the servo to
get there
}


void force()
{
  valf = analogRead(potpinf);
  Serial.println(map(valf, 0, 120, 0, 180));
  valf = map(valf, 0, 120, 500, 2400);
  totalf = totalf - readingsf[readIndexf];
  readingsf[readIndexf] = valf;
  totalf = totalf + readingsf[readIndexf];
  readIndexf = readIndexf + 1;
  if (readIndexf >= numReadingsf) {
    readIndexf = 0;
  }
  averagef = totalf / numReadingsf;
  myservo.writeMicroseconds(averagef);

  delay(2);
}

void stepp() {

  int brightness = analogRead(light_pin); // 0-1023
//  Serial.println(brightness);

  brightness = analogRead(light_pin);
  currStep = map(brightness, 0, 1023, 0, 2 *
stepsPerRevolution);

  myStepper.step(currStep-lastStep);
  lastStep = currStep;
  delay(100);
}


void pos() {


  //From IR to Encoder units through mapping for pid setpoint

  if (true) {
    temp = map(irRead(), 100, 500, 0, 360);

    if (temp > 179)
      temp = temp - 360;

    delay(10);
    loop_check = 1;
    enc_pos.write(0);
    while (loop_check) {
      pid_set = temp;


      Serial.print("setpoint (from IR) is ");
      Serial.println(pid_set);

      new_pos_val = ((int(enc_pos.read())) % 700) * float(360) /
700;


      Serial.print("Position from encoder = ");
      Serial.println(new_pos_val);
      //PID stuff
      pid_in = new_pos_val;
      error = pid_set - pid_in;
      error_d = error - error_last;
      error_last = error;
      Serial.print("Error = ");
      Serial.println(pid_set - pid_in);
      if ((error > 5) || (error < -5) && pid_set != 0) {


        error_i = error_i + error;
        pid_out = pid_p * error + pid_i * error_i +  pid_d *
error_d;


        if (pid_out < 0) {
          //   low on 2 and high on 1
          digitalWrite(pin_l1, HIGH);
          digitalWrite(pin_l2, LOW);
        } else {
          //   low on 1 and high on 2
          digitalWrite(pin_l1, LOW);
          digitalWrite(pin_l2, HIGH);
        }

        if (pid_out > 255)
          pid_out = 255;
        else if (pid_out < -255)
          pid_out = -255;


        analogWrite(enable_pin, abs(pid_out));
        Serial.print("Val from PID (PWM) = ");
        Serial.println(pid_out);
      }
      else
      {
        analogWrite(enable_pin, 0);
        digitalWrite(pin_l1, HIGH);
        digitalWrite(pin_l2, HIGH);
        // pid_set = 0;
        error_i = 0;
        error_d = 0;
        error = 0;
        loop_check = 0;
        enc_pos.write(0);
        //
Serial.println("OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO");
Serial.println("OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO");
        //
Serial.println("OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO");
        delay(1000);
      }
    }
  }

}

void vel() {

  //Check each second for change in position

  if (vcount > 100)
  {
    analogWrite(enable_pin, 100);
    vtemp = map(irRead(), 100, 500, 300, 450);
    //    vtemp = Serial.parseInt();
    //    vtemp = vtemp;
    //   verror_i = 0;
    //   verror_d = 0;
    //    time_int = millis();
    delay(10);
    vcount = 0;
    Serial.println("Triggered Triggered Triggered Triggered
Triggered Triggered Triggered Triggered Triggered Triggered ");
    analogWrite(enable_pin, map(vtemp, 300, 450, 200, 255));
    delay(2000);
  }
  else
  {
    vcount++;
    vpid_set = vtemp;
    encoder0Pos =  360 * double(enc_pos.read()) / 700.0;
    tmp_Pos = encoder0Pos;
    Serial.print("pt = ");
    ang_vel = -1 * float(1000 * (tmp_Pos - init_pos)) /
((millis() - time_int));

    if (ang_vel > 500) {
      ang_vel = 500;
    }
    if (ang_vel < -500) {
      ang_vel = -500;
    }
    Serial.println(ang_vel);

    time_int = millis();
    init_pos = tmp_Pos;

    //    if((ang_vel <0.6) && (ang_vel >-0.6))
    vpid_in = ang_vel;
    verror = vpid_set - vpid_in;

    if (verror > 50)
      verror_d = verror - verror_last;
    verror_last = verror;
    Serial.print("vpid_set = ");
    Serial.println(vpid_set);


    Serial.print("verror = ");
    Serial.println(vpid_set - vpid_in);
    Serial.print("vpid_in = ");
    Serial.println(vpid_in);


    //   if(vcount  ==1000) {
    //     verror_i = 5;     //By obversation for 30RPM
    //     vcount = 0;
    //   }
    if (abs(verror) > 50) {
      verror_i = verror_i + verror;
      vpid_out = vpid_p * verror + vpid_i * verror_i +  vpid_d *
verror_d;

      Serial.print("verror_i = ");
      Serial.println(verror_i);

      Serial.print("verror_d = ");
      Serial.println(verror_d);

      Serial.print("PID out = ");
      Serial.println(vpid_out);

      Serial.print("Enc h = ");
      Serial.println(360 * double(enc_pos.read()) / 700.0);

      if (vpid_out > 0) {
        //   low on 2 and high on 1
        digitalWrite(pin_l1, HIGH);
        digitalWrite(pin_l2, LOW);
      } else {
        //   low on 1 and high on 2
        digitalWrite(pin_l1, LOW);
        digitalWrite(pin_l2, HIGH);
      }

      if (vpid_out > 255)
        vpid_out = 255;
      else if (vpid_out < -255)
        vpid_out = -255;

      if (vpid_set > 0 && vpid_out < 0) {
        vpid_out = 0;
      }
      if (vpid_set < 0 && vpid_out > 0) {
        vpid_out = 0;
      }

      analogWrite(enable_pin, abs(vpid_out));
      Serial.print("Val from PID (PWM) = ");
      Serial.println(vpid_out);
      //      delay(50);

    }
  }

}
void loop() {
  int reading = digitalRead(buttonPin);

  // check to see if you just pressed the button
  // (i.e. the input went from LOW to HIGH),  and you've waited
  // long enough since the last press to ignore any noise:

  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer
    // than the debounce delay, so take it as the actual current
state:

    // if the button state has changed:
    if (reading != buttonState) {
      buttonState = reading;
      // only toggle the LED if the new button state is HIGH
      if (buttonState == HIGH) {
        sysstate = (sysstate + 1) % 5;
      }
    }
  }
  lastButtonState = reading;

  switch (sysstate)
  { case 0:
      Serial.print("frc");
      force();
      break;
    case 1:
      portential();
      //        Serial.print("portential");
      break;
    case 2:
      Serial.println("stp");
      stepp();
      break;
    case 3:
      Serial.print("pos");
      pos();
      break;
    case 4:
      Serial.print("vel");
      vel();
      break;
    default:
      break;
  }
  delay(2);

}

// Take multiple readings, and average them out to reduce false
readings
int irRead() {
  int averaging = 0;             //  Holds value to average
readings
  // Get a sampling of 10 readings from sensor
  for (int i = 0; i < 10; i++) {
    distance = analogRead(irSense);
    averaging = averaging + distance;
    delay(55);      // Wait 55 ms between each read
    // According to datasheet time between each read
    //  is -38ms +/- 10ms. Waiting 55 ms assures each
    //  read is from a different sample
  }
  distance = averaging / 10;      // Average out readings
  if (distance > 500) {
    distance = 500;
  }
  if (distance < 100) {
    distance = 100;
  }
  Serial.println("Triggered Triggered Triggered Triggered
Triggered Triggered Triggered Triggered Triggered Triggered ");
  return (distance);
}
```