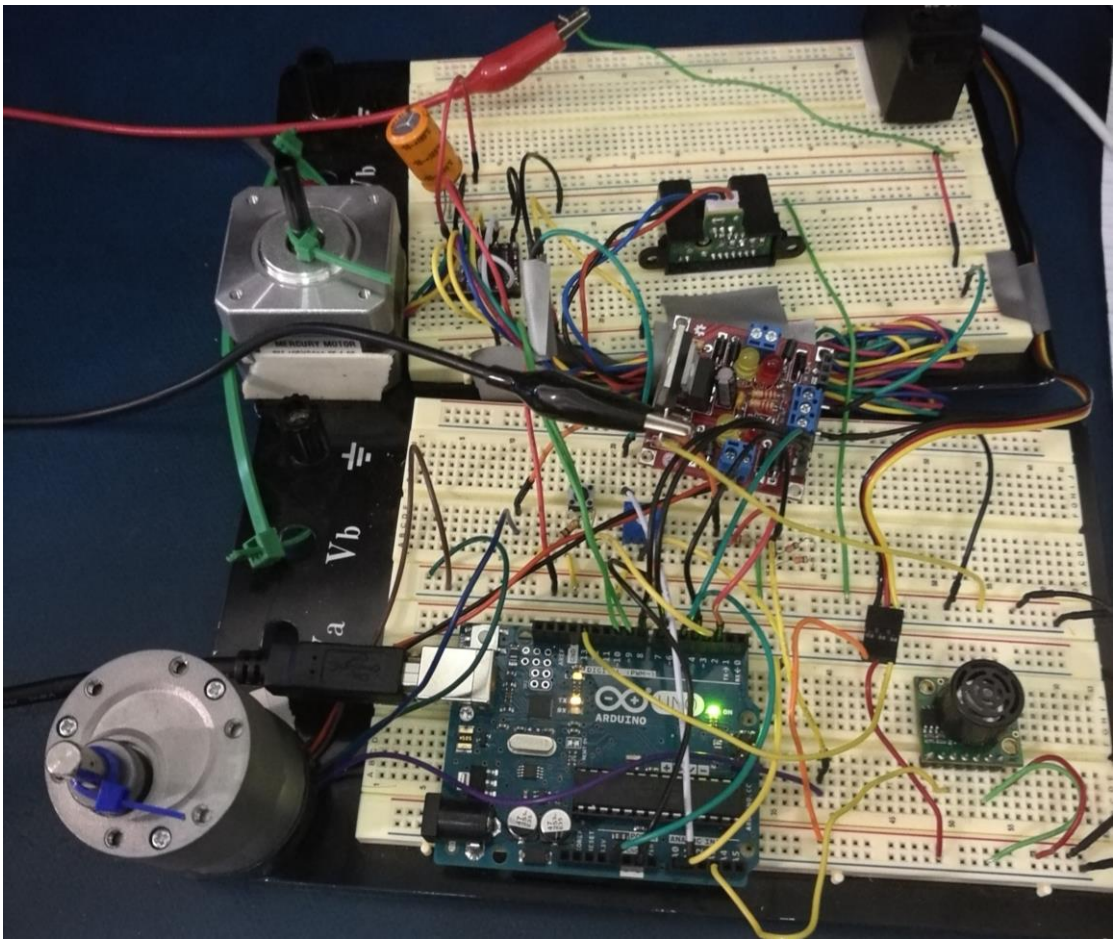
Team F: Rescue Rangers

Juncheng Zhang

Teammates: Karthik Ramachandran
Sumit Saxena
Xiaoyang Liu



ILR01

10/14/2016

1、Individual Progress

1.1、Overview

For Sensors and Motor Lab, my primary role was for controlling DC motor and integration of potentiometer. Also, I helped build the hardware system and integrate all codes from different parts, especially for the DC motor part.

More specifically, my time is mainly spent on the following work:

1) Solder and test the Solarbotics L298 Compact Motor Driver Kit(Driver IC)
2) Control the DC motor based on potentiometer value(Sensor-input control)
3) Control the velocity of DC motor using PID(Velocity control)
4) Control the position of DC motor using error correction method(Position control)

Details of implementation will be discussed below.

1.2、Driver IC

To provide enough circuit for DC motor and use PWM to control the speed of DC motor, the L298 motor driver IC is used. However, L298 is very hard to use, so a compact motor driver kit with other additional functions could be very helpful in our case, which is shown in Figure1.

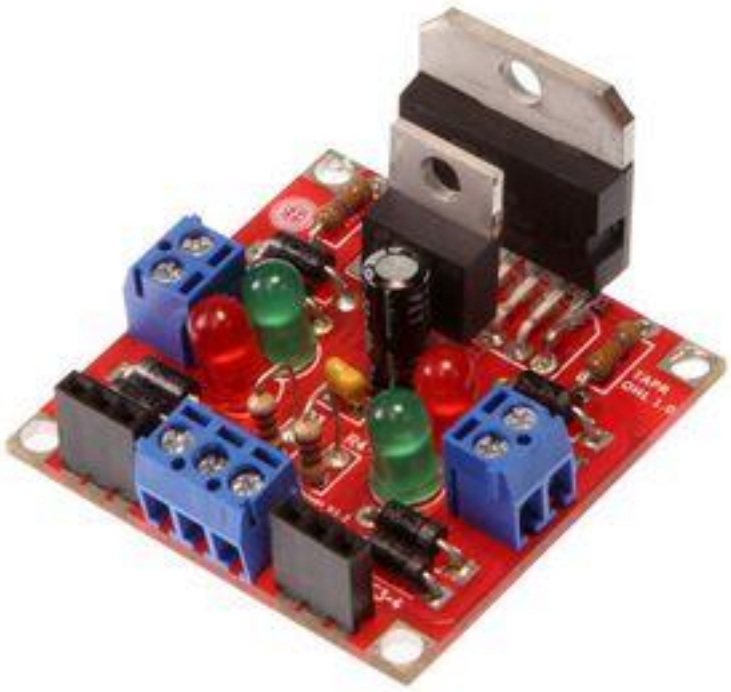


Figure1 Solarbotics L298 Compact Motor Driver Kit

After constructing the circuit, I ran a test to see whether the 5V regulator and the L298 motor driver IC can operate normally. I used three jumper wires, two wires were connected from opposite polarities to the input lines (one from '+5V', one from '-'). Then, I ran a third jumper from the '+5' to 'E1-2' to turn it on (LED lights up), and moved the third jumper from

‘+5’ to ‘-’ to turn the LED off.

To control the DC motor using microcontroller and this motor driver kit, the logic table needs to be understood. From figure2 below, we can change the rotation direction of motor by controlling L1、L2, and control the velocity by giving PWM waveforms with different duty cycles to the Enable pin.

LOGIC TABLE

Enable	L1	L2	Result
L	L	L	OFF
L	L	H	OFF
L	H	L	OFF
L	H	H	OFF
H	L	L	BRAKE
H	L	H	FORWARD
H	H	L	BACKWARD
H	H	H	BRAKE
H	L	L	BRAKE
H	PWM	H	FWD-SPD
H	PWM	L	BCK-SPD
H	H	H	BRAKE

Enable	L3	L4	Result
L	L	L	OFF
L	L	H	OFF
L	H	L	OFF
L	H	H	OFF
H	L	L	BRAKE
H	L	H	FORWARD
H	H	L	BACKWARD
H	H	H	BRAKE
H	L	L	BRAKE
H	PWM	H	FWD-SPD
H	PWM	L	BCK-SPD
H	H	H	BRAKE

Figure2 Logic table for motor control

1.3、Velocity Control

Reading sensor values through analogRead and changing velocities through analogWrite PWM outputs with different duty cycles are not difficult at all. The tricky part is that since the motor velocity does not change linearly based on input voltages, it is difficult to make the DC motor run in a certain given speed. One way to control the speed is PID controller.

PID controller requires to know the current speed to calculate the error in terms of the desired speed, which makes attaining velocity information through encoder necessary. The encoder for our DC motor has 180 steps per turn and 2 channels showing the current state of the encoder. Below is an image showing the waveforms of the A & B channels of an encoder.

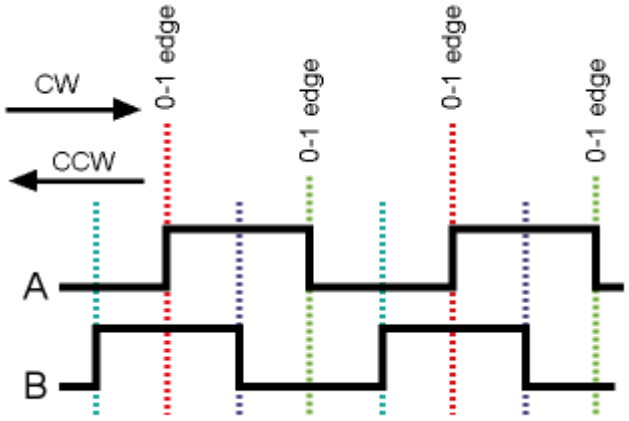


Figure3 Channel waveforms of the encoder

To determine the rotation direction of the motor, I attach the falling edge of channel A as

an interrupt to Arduino board. When channel A jumps from high level to low level, an interruption function is called, which will check the voltage of channel B. If channel B is low, the motor rotates clockwisely. Otherwise, the rotation direction would be anticlockwise. In order to get the motor velocity, I simply use a 1-second timer to calculate it based on step changes in one second.

For PID controller, its process can be illustrated in Figure4, a block diagram of PID controller in a feedback loop.

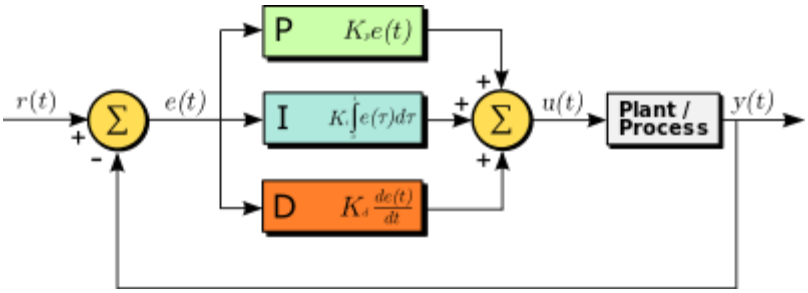


Figure4 Block diagram of PID controller

For the implementation of PID controller, I used the Arduino PID Library with lots of build-in functions. It saved me a lot of time writing my code, and simplified my work to tune the PID parameters. The final parameter selection is like: Kp=4, Ki=3, Kd=0.

1.4、Position Control

Precise position control for DC motor can be very complicated due to the inertia of the motor. My strategy to deal with inertia is actually very simple and straightforward, which is simply counting error steps for rotating different degrees. For example, in experimental phase, if the motor is asked to rotate 90 degree, I will not stop the motor until the encoder tells me that it achieves the destination. In consideration with the inertia, the motor will still move forward several steps even the voltage provided is already 0, which results in some error. If the error of rotating 90 degree is 10 degree, my method is to amend the position where the motor stops. Specifically, the motor will be asked to stop when it rotates 80 degree, and is possible to finally stop at 90 degree.

This method cannot eliminate the error, because the motor may have different speeds at 80 degree and 90 degree, which means the error steps are different. To explain it in another way, the motor may just move forward 5 degree when it stops rotating at 80 degree. As a result, the motor stops at 85 degree instead of 90 degree, which means 5 degree error still exists. Actually, a better way to eliminate error is to add moderating process, which gradually slow down the motor before it stops. This will definitely decrease the error, but I don’t experiment with this method because of limitation of time.

2、Challenges

The primary challenge I faced during the lab was letting my code work during the code integration. Before I gave my code to Karthik, who was responsible for software integration, my 3 different functions for DC motor all worked well, including sensor-input control, velocity control and position control. However, only sensor-input control worked when my code was put into the integrated one. After using Serial.println to test different functions, I realized that my 1-second timer function was never called when the code for controlling servo motor was added. This phenomenon is perhaps because that there are a lot of delay functions in servo motor function, which are necessary for servo motor, but may influence the function of my timer. My solution is to move the original code in 1-second timer to other functions. Decisions about the place where certain codes are moved, actually are analyzed carefully, in order to keep the whole system working with minimal influence.

Another problem during the integration process was the encoder always showed that the motor rotated forward regardless of its actual direction. To solve this question, I first used serial.println to test different functions, but still did not figure out the reason. Then, I uploaded my code only for DC motor to arduino instead of the integrated one, just to assure that this abnormal behavior of encoder was not caused by the code integration. However, even I used my original code, which was proved to be correct, the problem still remained, which means that there is something wrong with the hardware, not the software. After clearly checking the wire connection, I found that one channel of the encoder was not inserted appropriately into the breadboard. This mistake caused that the voltage level of this channel was always low, which makes encoder unable to distinguish the correct rotation direction of the motor. After I fixed this bug, the whole integrated system finally worked.

3、TeamWork

This project necessitated a well-planned distribution of work and integration phase. After a team meeting to discuss the project, we broke the work down as follows:

Table1 Work distribution form

Member	Work
Karthik Ramachandran	Servo Motor, Ultrasonic Sensor, Software Integration
Sumit Saxena	GUI Design, Transmission Protocol for Serial Data
Juncheng Zhang	DC Motor, Potentiometer, Pressed Button Function
Xiaoyang Liu	Stepper Motor, Infrared Proximity Sensor, Hardware Integration

The team worked with great coordination during execution of the entire task. Before anyone started working on their respective Arduino codes, Sumit already designed the transmission protocol for serial data so that GUI could send valuable commands to control each motor. Also, Karthik asked us to use Struct to store the status of motor, which is really helpful for the final code integration. Everyone completed their part well before time and integration went smoothly

4、Future Plans

From now until the next ILR, there are only 7 days. Our main focus is on finalizing types and models of sensors to be used in our projects. We will search the working altitude、resolution、area coverage, as well as allowable moving velocity of different RGB cameras、thermal cameras and sound sensors respectively, and try to find the most suitable one to our autonomous aerial system for search and rescue. I will take charge of choosing RGB camera, Sumit will find appropriate thermal camera, and Karthik will be responsible for sound sensor. After we select sensors separately, we will try to design initial strategy for local navigation pattern according to capabilities of different sensors.

At the same time, Xiaoyang will get familiar with DJI SDK in this week, so that we can work on navigation part as soon as possible. Furthermore, Xiaoyang and I are in the same team of Computer Vision project, and we plan to develop a human detection algorithm, which might be applied into our MRSD project and could be helpful to capture human signature when the drone searches the area. We have finished the project proposal this week and am going to search related literature and learn more about OpenCV in a next few weeks.

5、Reference

Figure1: https://solarbotics.com/product/k_cmd/
Figure2: https://cdn.solarbotics.com/products/documentation/kcmd_manual_v1-4.pdf
Figure3: http://playground.arduino.cc/Main/RotaryEncoders
Figure4: https://en.wikipedia.org/wiki/PID_controller

6、Appendix

```
#include "TimerOne.h"
#include "PID_v1.h"

// Pin Definition
#define encoder0PinA    2
#define buttonPin 3
#define encoder0PinB    4
#define E1_2Pin 5
#define l1Pin 8
#define l2Pin 7

void DC_Initial();
void DC_Move_Degree(int degree, int d);
void DC_Move_PID(int velocity, int d);
extern void callback1();

// Initialization for Interrupt
long debounce_time=150;        // the debounce time; increase if the output flickers
volatile unsigned long last_micros;

//Initialization for motr
int encoder0Pos;
int currentLoc=0;
int lastLoc=0;
double vel; //velocity for encoder
bool onoff; //shows that motor is moving or not, true means motor is moving.

int state=3; //state of the motor, 0 is for motionless, 1 is for sensor input, 2 for is pid control,3 is for position control
int dir=LOW; //direction of the motor, LOW means forward

//Define global variables for state1
int sensorPin = A0;        // select the input pin for the potentiometer
int sensorValue = 0;    // variable to store the value coming from the sensor
int sensorSpeed=0;      // Speed sent to motors according to the sensor

//Define global variables for state2, i.e. DC_Move_PID
```

```
double Setpoint, Input, Output;
double Kp=4, Ki=3, Kd=0;
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

//Define global variables for state3, i.e. DC_Move_Degree
int destLoc;
int error; //for position amendment
bool DC_Move_Start=false;    // to make sure DC moves only if it is required

void setup()
{

   DC_Initial();
}

void loop() {
   switch(state)
   {
      case 0:
         analogWrite(E1_2Pin, 0);
      break;
      case 1:
         sensorValue = analogRead(sensorPin);
         sensorSpeed = map(sensorValue, 0, 1023, 0, 255);
         analogWrite(E1_2Pin, sensorSpeed);
      break;
      case 2:
         DC_Move_PID(60, 1);
      break;
      case 3:
          if(DC_Move_Start==false)
          {
             //Serial.println("GoGoGo!");
             DC_Move_Degree(360, 1);
          }
      break;
      default:
      break;
   }
```

```
    delay(5);
}

//this function is called every 1 second
void callback()
{
    currentLoc=encoder0Pos;
    vel=((double)(currentLoc-lastLoc))/3;
    lastLoc=currentLoc;
    if(dir==1)
    vel=-vel;
    Serial.print("current state is:    ");
    Serial.println(state);



    Serial.print("current speed:    ");
    Serial.println(vel);

  // Serial.println(destLoc);
    if(state==2)
    {
        myPID.SetMode(AUTOMATIC);
    }
    else
    {
        myPID.SetMode(MANUAL);
      }

}

//this is for button interruption
void debounceInterrupt()
{
    if((long)(micros()-last_micros)>=debounce_time*1000)
    {
        Interrupt();
        last_micros=micros();
    }
}
```

```
void Interrupt()
{

   if(state==1)
   {
      dir=!dir;
      digitalWrite(l1Pin,dir);
      digitalWrite(l2Pin,!dir);
   }
}

//this is for encoder interruption
void debounce_Encoder() {
     //delay(10);
       //Serial.println("encoder!");
       if (digitalRead(encoder0PinB) == LOW)
       {    // check channel B to see which way
          encoder0Pos = encoder0Pos - 1;                // CCW
       }
       else
       {
          encoder0Pos = encoder0Pos + 1;                 // CW
       }
       //currentLoc=encoder0Pos;
       Serial.println(encoder0Pos);
       if(state==3)
       {
          if(DC_Move_Start==true)
        {
             if(dir==LOW)
             {
                if((encoder0Pos+error)>=destLoc)
                {
                     DC_Move_Start=false;
                  analogWrite(E1_2Pin, 0);
                  state=0;      //let it stop
                }
             }
             else
```

```
            {
               if((encoder0Pos-error)<=destLoc)
               {
                    DC_Move_Start=false;
                  analogWrite(E1_2Pin, 0);
                  Serial.println("stop");
                  state=0;    //let it stop
               }
            }
         }
      }
  // Serial.println (encoder0Pos, DEC);                  // debug - remember to comment out
}


void DC_Initial()
{
      //Timer1 Initialization
   Timer1.initialize(1000000);
   Timer1.attachInterrupt(callback);    // attaches callback() as a timer overflow interrupt

//Initialize Input
   pinMode(encoder0PinA, INPUT);
   digitalWrite(encoder0PinA, HIGH);             // turn on pullup resistor
   pinMode(encoder0PinB, INPUT);
   digitalWrite(encoder0PinB, HIGH);             // turn on pullup resistor
   pinMode(buttonPin,INPUT);


//Initialize Output
   pinMode(l1Pin, OUTPUT);
   pinMode(l2Pin, OUTPUT);
   pinMode(E1_2Pin, OUTPUT);

//Initialize the Interrupt functions
   attachInterrupt(digitalPinToInterrupt(3),debounceInterrupt,FALLING);
   attachInterrupt(digitalPinToInterrupt(encoder0PinA ), debounce_Encoder, FALLING);    //
encoder pin on interrupt 0 - pin 2
```

```
    encoder0Pos=0;
    digitalWrite(l1Pin,dir);
    digitalWrite(l2Pin,!dir);
    analogWrite(E1_2Pin, 0);

    Serial.begin (9600);
    Serial.println("start");                          // a personal quirk


      //initialize the pid value
    Input = vel;
    Setpoint = 0;
    myPID.SetMode(MANUAL);
    }

// for state2 pid control, i.e. velocity control
  void DC_Move_PID(int velocity, int d)
    {
          dir=!d;
          Input = vel;
          Setpoint = velocity;
          myPID.Compute();
          digitalWrite(l1Pin,dir);
          digitalWrite(l2Pin,!dir);

          analogWrite(E1_2Pin, Output);
    }



// for state3 motor control, i.e. positon control
    void DC_Move_Degree(int degree, int d)
    {
       DC_Move_Start=true;
       int increStep=floor(degree/2);
       if(degree>=150)
       {
          error=30;
```

```
    }
    else if(60<=degree<150)
    {
       error=15+(degree-60)/6;
    }
    else
    {
         error=(int)degree/4;
     }
    if(d==1)
    {
       destLoc=encoder0Pos+increStep;
       dir=LOW;
    }
    else
    {
       destLoc=encoder0Pos-increStep;
       dir=HIGH;
    }
     digitalWrite(l1Pin,dir);
     digitalWrite(l2Pin,!dir);
     analogWrite(E1_2Pin, 140);
}
```