Carnegie Mellon University

Team B : AutoPirates

ILR1: Motor and Sensor Lab

Author:
Shiyu Dong

Team Member:
Tushar Chugh
Tae-Hyung Kim
Bikramjot Hanzra
William Seto

October 16, 2015

1. Individual Progress

For this Sensor and Motor Lab, I implemented the force resistive sensor and worked with William Seto to implement the PID control algorithm for both position control and velocity control for the DC motor.

1.1 Force resistive Sensor

The force resistive sensor I implemented is shown as figure 1. The force resistive sensor measures the pressure it's being applied to the sensing area. With the increase of force, the resistance of the sensor decreases.

Figure 1: Force resistive Sensor

Connect force resistive sensor to Arduino as figure 2 shows. Select RM = 10KΩ. According to the datasheet of the sensor, the transfer function is shown as figure 3.

Figure 2: Circuit

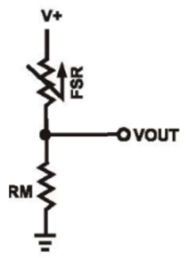
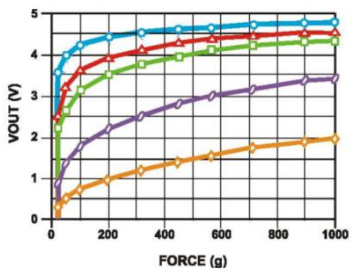


Figure 3: Transfer Function



The Pin *vout* of circuit in figure 2 is connected to Pin *A0* in Arduino. In Arduino, the voltage of the sensor can be read using *analogRead(A0)*. Hence the resistance of the sensor can be calculated using the following equation:

$$V_{out} = \frac{R_M V_{cc}}{R_{M+R_s}}$$

It is hard to measure the force we applied to the sensor, so we didn't measure and plot the transfer function of the force resistive sensor. Arduino will send the force sensor data to GUI and GUI will be able to display the real-time value.

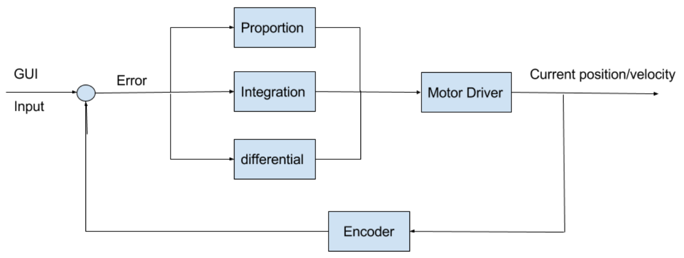
1.2 Position Control of DC Motor

For the position control of the DC motor, I was controlling the relative position of the shaft. That is, the shaft of the motor will move a certain angle according to user input. And the position feedback is given by the encoder.

To begin with the control, the first thing I did was to figure out the output for the encoder. Since we used a two-channel encoder, we were able to get positive and negative direction ticks separately. For every one-degree move of the shaft, there is a one-pulse output from the encoder from the corresponding channel. When the interrupt is attached to both channels, each tick from the encoder will trigger interrupt and will be summed up in the interrupt, making shaft angle measurement possible.

The closed loop PID control is based on the feedback of the position. The block diagram is showed as figure 4. The system receives input from GUI and compares the desired position with current position, and gets the error. The PID control algorithm is to calculate the proportion, integration and differential of the error and tune the gain for each part to get best performance, for example, the least overshoot and settling time.

Figure 4 PID Control Block Diagram



The programming implementation and function of PID algorithm is shown as Table 1.

Part	**Programming Implementation**	**Function**
P	*desired_value - encoderPos*	Decrease rise time, settling time and steady-state error
I	*i_error + p_error*	Eliminate steady-state error
D	*p_error - pre_error*	Decrease overshoot and settling time

Table 1 PID function and implementation

To tune the PID parameter for DC motor, I first set K_i and K_d to 0 and increased K_p until the shaft of the motor becomes unstable. Then chose a reasonable gain for proportional part such as $0.5K_p$. At this time, there was still a relatively big overshoot. So I increases K_d to decrease the overshoot. The serial port monitor or the GUI are able to display the feedback of encoder, i.e. the current position. So if there is an error between final steady-state position and desired position, I can increase K_i to eliminate the steady-state error. It's important to specify here that in order to make the system stable, the value of K_i cannot be set too high.

1.3 Velocity Control of DC Motor

The velocity control of DC motor is nearly the same as position control. The only difference is the method to get velocity feedback based on the encoder. My final implementation is to set a delay of 50*ms* in the loop of velocity control and get the difference of the encoder position. So we defined a velocity:

$$V = encoderPos(50) - encoderPos(0)$$

Because we didn't divide the difference by time. So the unit of the *V* we defined is actually degree per 50*ms*.

2. Challenges

The most challenging part is in the implementation of velocity control. I have tried three different way to get the velocity feedback and finally chose the 3rd one.

The first method I used is to get the difference of the shaft position without a delay in the control loop. This method proved to have a great error because the sampling time was too short so the difference was mostly 0, 1 or 2. In this kind of situation. It was very inaccurate because a simple noise will cause great change in the feedback.

The second method I used is to divide the difference I got from method 1 by the elapsed time. But this is still not an accurate feedback. For example, if the elapsed time is 20*ms*, so I have to actually multiply 50 to get the speed. Then the difference will be 0, 50 or 100, and the same error will show on the GUI, which is not what we expected.

The third method, our final choice, is to set a delay time in the control loop, like 50*ms*, and then calculate the difference of shaft position in an interval of 50*ms*. With the increase of sampling time, the difference is more accurate and not be easily influenced by noises.

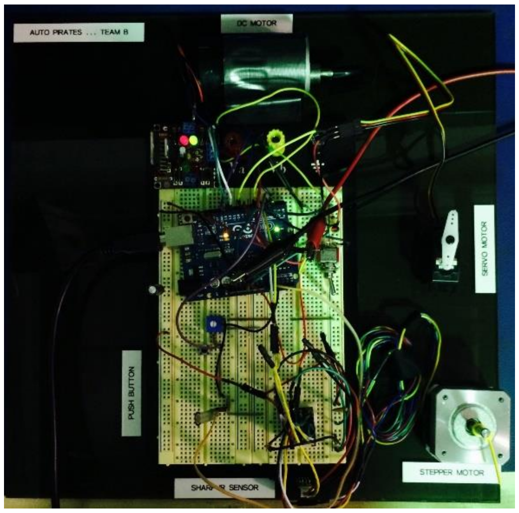
But this method has an obvious weakness - It relies on the accurate time of the delay. So when we integrated the code together, the delay time changed, therefore we needed to retune the PID parameters.

3. Teamwork

Each team member was in charge of implementing a sensor and control the motor. The final diagram of the system is shown as figure 5.

- Tushar Chugh: Tushar worked on the designing the Graphical User Interface(GUI) , sending and receiving data in a certain format through serial port. In addition, he integrated GUI with Arduino, including the pushbutton on breadboard that is used for changing modes in GUI.
- Bikramjot Hanzra: Bikramjot integrated toggle switch and servo motor. And he also worked on plotting the transfer function of Sharp IR Sensor using Matlab.
- Tae-Hyung Kim: Tae-Hyung implemented potentiometer and stepper motor with Arduino.
- William Seto: William implemented the Sharp IR Sensor and also helped me with the position control of DC motor. Besides, he also integrated all the separate codes in Arduino.

Figure 5 Motor and Sensor System



4. Future Plan

This motor and sensor lab is a good experience to working with teammates and integrating systems. But since our project is mostly software-based, I will work on the project in the future.

My job for the project is mostly in perception. Now we have successfully integrated an open source library called OpenCPN with a radar plugin. Based on the plugin, we can receive data from the radar and plot the graph on the screen.

The next step would be successfully compile the source code of OpenCPN with our own modification so that we can log and play data of radar. In addition to this, I should start reading papers in the area of radar perception and implement some algorithms once we successfully log the data.

Since our first field test will start in the early November, it is also important to make a detailed test plan to help us get a productive field test.

5. Reference

Force Resistive Sensor Datasheet: http://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/2010-10-26-DataSheet-FSR400-Layout2.pdf

Appendix: Arduino Source Code for my work

```
#define InA1          11                   // INA motor pin

#define InB1          12                   // INB motor pin

#define PWM1          10                   // PWM motor pin


#define FORWARD        1                   // direction of rotation

#define BACKWARD       2                   // direction of rotation

#define force_data    A1

#define encoderPinA 2

#define encoderPinB 3


volatile int encoderPos = 0; //gives tick position from 0 reference

volatile int last_encoder_pos = 0;

volatile float velocity = 0.0;


/* POSITION PID CONSTANTS */

volatile float p_error=0;    //current error

volatile float pre_error=0;   //previous p error

volatile float d_error=0;    // current p error - previous p error

volatile float i_error=0;    // i error, integrate p errors

const float kp          =6; //P parameter

const float ki          =0.001; //I parameter

const float kd          =9; //D parameter

/**************************/


/* VELOCITY PID CONSTANTS */
```

```
volatile float p_error_vel  =0;     //current error

volatile float pre_error_vel=0;     //previous p error

volatile float d_error_vel  =0;     // current p error - previous p error

volatile float i_error_vel  =0;     // i error, integrate p errors

const float kp_vel          =25;  //P parameter

const float ki_vel          =0.4; //I parameter

const float kd_vel          =5;  //D parameter

float p_value = 0.00;

float pp_value = 0.00;

float ppp_value = 0.00;

float force_average = 0.00;

volatile int Motor_speed_value=0;

const int max_speed=255;


void setup() {

  pinMode(InA1, OUTPUT);

  pinMode(InB1, OUTPUT);

  pinMode(PWM1, OUTPUT);

  pinMode(encoderPinA, INPUT);

  pinMode(encoderPinB, INPUT);

  digitalWrite(encoderPinB, HIGH);       // turn on pullup resistor

  digitalWrite(encoderPinA, HIGH);

  attachInterrupt(digitalPinToInterrupt(encoderPinA), read_encoder, CHANGE);

  // encoder pin on interrupt 0 (pin 2)


  Serial.begin (115200);

}


void loop() {

  float force_value  = analogRead(force_data);


  //example

  int mode = 2; //1 -> speed control ; 2 -> position control
```

```
 switch (mode) {
  case 1:
   speedControlDC(5);
   break;
  case 2:
   positionControlDC(90);
   break;
 force_value = 0.25*(force_value + p_value + pp_value +ppp_value);

 Serial.println(force_value);
 ppp_value = pp_value;
 pp_value = p_value;
 p_value = force_value;

 delay(100);
 }

 //Serial.print("pwm value: "); Serial.print(Motor_speed_value);
 /*
 Serial.print(" encoder: "); Serial.print(encoderPos);
 Serial.print(" error: "); Serial.print(p_error);
 Serial.print(" pre_error: "); Serial.println(pre_error);
 */
 /*
 Serial.print(" velocity: "); Serial.print(velocity);
 Serial.print(" vel error: "); Serial.print(p_error_vel);
 Serial.print(" vel pre_error: "); Serial.println(pre_error_vel);
 */


}

void positionControlDC(int desired_degrees) {
 while(1) {
    positionPID(desired_degrees);

    run_motor();

    delay(50);

  }

}


void speedControlDC(int desired_velocity) {

  while(1) {

    velocityPID(desired_velocity);

    run_motor();

    delay(50);

  }

}


int positionPID(int desired_value) {

  p_error = desired_value - encoderPos;

  d_error = p_error - pre_error;

  i_error = i_error + p_error;


  pre_error = p_error; //store current error


  Motor_speed_value = kp*p_error + ki*i_error + kd*d_error; //PID calculation


  if (Motor_speed_value>=max_speed) {

    Motor_speed_value = max_speed;

  }

  else if(Motor_speed_value<=-max_speed) {

    Motor_speed_value = -max_speed;

  }


}


void velocityPID(int desired_value){
  velocity = encoderPos - last_encoder_pos; // counts per ~50ms,

  last_encoder_pos = encoderPos;


  p_error_vel = desired_value - velocity;

  d_error_vel = p_error_vel - pre_error_vel;

  i_error_vel = i_error_vel + p_error_vel;


  pre_error_vel = p_error_vel; //store current error


  Motor_speed_value = kp_vel*p_error_vel + ki_vel*i_error_vel + kd_vel*d_error_vel;//PID calculation


  if (Motor_speed_value>=max_speed) {

    Motor_speed_value = max_speed;

  }

  else if(Motor_speed_value<=-max_speed) {

    Motor_speed_value = -max_speed;

  }


}



/* ENCODER INTERRUPT */

void read_encoder(){


  if (digitalRead(encoderPinA) == HIGH) {   // found a low-to-high on channel A

    if (digitalRead(encoderPinB) == LOW) {  // check channel B to see which way

                                             // encoder is turning

      encoderPos = encoderPos + 1;         // CW

    } else {

      encoderPos = encoderPos - 1;         // CCW

    }

  }

  else                                     // found a high-to-low on channel A

  {
    if (digitalRead(encoderPinB) == LOW) {   // check channel B to see which way
                                             // encoder is turning
      encoderPos = encoderPos - 1;         // CCW
    } else {
      encoderPos = encoderPos + 1;         // CW
    }
  }
}

void run_motor(){
  if(Motor_speed_value>=0) {
    analogWrite(PWM1, Motor_speed_value);
    digitalWrite(InA1, LOW);
    digitalWrite(InB1, HIGH);
  } else {
    analogWrite(PWM1, -Motor_speed_value);
    digitalWrite(InA1, HIGH);
    digitalWrite(InB1, LOW);
  }
}
```