Carnegie Mellon University

16-681A

MRSD Project I

Individual Lab Report 01
Team C: COBORG

Author: Gerald D'Ascoli

Remaining Team C Members:
Jonathan Lord-Fonda | Yuqing Qin | Husam Wadi | Feng Xiang

Sponsor:
Biorobotics Lab

February 25, 2021

Table of Contents

1 Individual Progress

1.1 Sensors & Motors Lab

My responsibilities for this lab were to design the PID controller for the DC motor, the stepper motor control with the ultrasonic sensor input, and to debug electrical circuit issues. The entire circuit can be seen in Fig 1, with the DC motor and Stepper motor labeled on the left, and the IR sensor and ultrasonic sensors labeled on the right.

For the stepper motor to be controlled by the ultrasonic sensor, it took moving average filter to stabilize the values. I also had to try a couple different ultrasonic sensors because some produced less consistent values while others were just broken due to previous misuse. I wrote a test script for the stepper motor to ensure functional control of the stepper motor in both directions based on the position read in by the ultrasonic sensor. Because of the significant upper limit of the range of the ultrasonic sensor and the increase in inaccuracies after a certain range, I designed the ultrasonic sensor input value to have an absolute cap at 100cm. With this cap, I could map the distance in cm from 0-100 of the ultrasonic sensors to the steps 0-400 of the stepper motor. From here it was just articulating the stepper motor from its current step to its desired step by changing the direction pin voltage to the needed value then looping through the step input pulses until the desired step is achieved. This script was simply test code as proof of functionality for the stepper controlled by the ultrasonic sensor, Jonathan then took the concept and re-wrote the code to allow it to be more efficiently implemented in the loop function with the other 2 motors. See code in Appendix A under "//Ultrasonic Control of Stepper".

Electrical debugging required continuity checks to ensure proper connectivity, solving common grounding issues, analyzing signals on the oscilloscope, and switching out to more stable power supplies for the Arduino to ensure signals would be delivered as strong and denoised as possible.

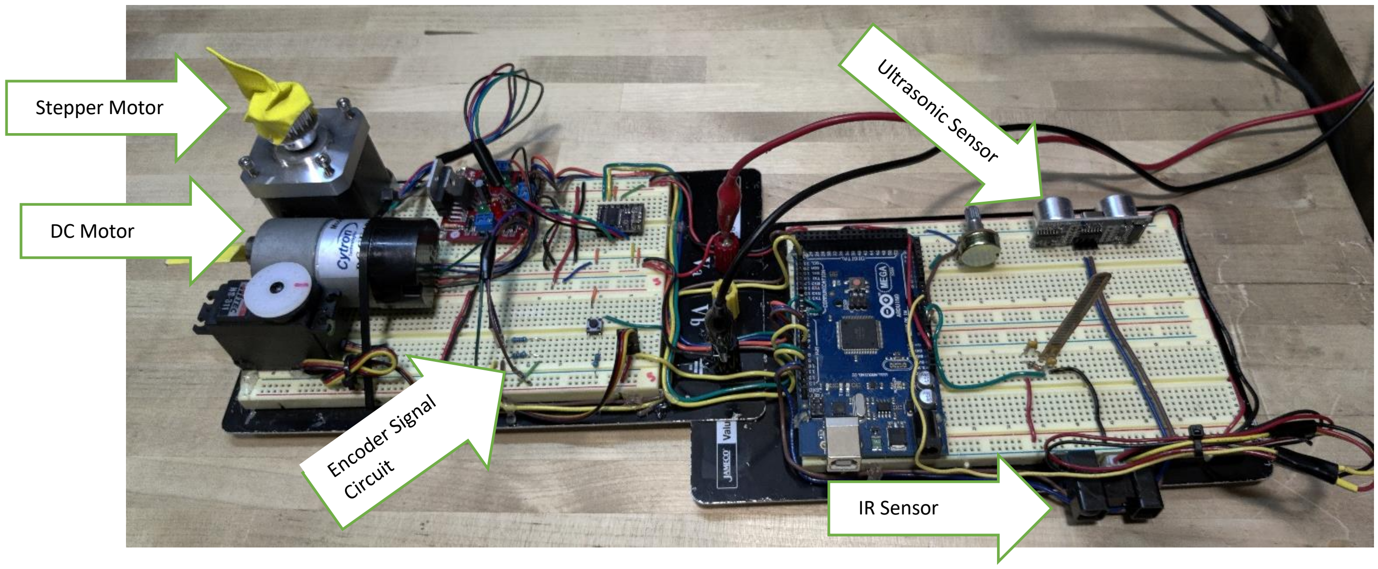

Fig 1. Full Sensors & Motor Lab Circuit Layout

1.1.1 DC Motor/IR Sensor

For control of the DC motor with the IR sensor, I had to design the PID controller to control the DC motor position given the IR distance sensor as input and the DC motor encoder to determine the state. Reading in the pure output of the IR sensor was far too noisy to use as an accurate input so it required an averaging filter to stabilize. I used a window size of 5 for this as it seemed to stabilize the value enough while still allowing for significant chances to be read in quickly. To read the output from the encoder for the state of the motor position for the PID, I had to implement a circuit with two 1kΩ resistors as pull-up resistors between the signal wire and the logical 5V line as shown in Fig 1 in order to stabilize the waveforms to make them digitally readable.

The Arduino code to implement the PID was accomplished using the stand Arduino PID library. The PID controller read in the input as the position of the DC motor in the form of degrees from 0 to 360. This was calculated by attaching an interrupt pin to the falling edge of encoder channel A and using that interrupt as a trigger of position change (only the A channel was used because we originally had an encoder with a broken B channel, this is further described in the "Challenges" section). This trigger either incremented or decremented a "count" variable depending on the current direction that the motor was spinning, which was determined by which motor pin was presently held HIGH digitally. This count position was translated into degrees by multiplying it by (360/counts per cycle) where counts per cycle was determined by a calibration loop (commented out in the given code) and was normally around 170-180. The setpoint for the PID controller was calculated based on the reading from the IR sensor. The IR sensor value was read in on an analog pin, run through the averaging filter for smoothing, then translated to a distance in centimeters given the equation on the sensor datasheet. To determine the setpoint for the PID, I took the accurate range of the IR distance sensor in cm to be 20-150cm and mapped that to 0 to 360 degrees to give desired position value in degrees to the PID. The output of the PID was designed to give direction and speed to the motor via the sign of the output value for the direction and a value from 0-255 for the speed. This was accomplished with the "myPID.SetOutputLimits(-255,255);" line of code. To implement the PID as the controller, I set the setpoint and the input to the PID, calculated the output value, then actuated the DC motor accordingly. If the output was positive, it meant the desired position was counterclockwise of the current position, so the digital pins to the H-bridge were set accordingly, and vice versa for a negative output value setting the direction to clockwise. To control the speed of the motor with the PID output value, I wrote the magnitude of the output value as a PWM signal to the H-bridge enable wire which set the speed to maximum at 255 and reduced speed from there. By design, as the current position approached the desired value, the output magnitude reduced therefore reducing the duty cycle of the PWM signal and slowing down the motor. I then tuned the Kp, Kd, and Ki gains to reduce oscillatory behaviors and converge to the desired value as quickly and stably as possible. The Arduino code can be found in Appendix B under "// Encoder PID".

1.2 COBORG Project

As Technical Lead on the Voice Subsystem for COBORG, I have made good progress on the voice recognition portion of our project. To date, I have developed and customized a standalone PocketSphinx node with custom libraries and dictionaries to tailor the voice recognition to the requirements of the COBORG. I have since been working on experiments to calculate the accuracy of the command recognition to determine if it meets the COBORG requirements. If I will have to start building a custom audio library among other methods of improving command recognition accuracy. Additionally, I have been working towards integrating the PocketSphinx implementation to our overall COBORG ROS map as the voice_node.

2 Challenges

2.1 Sensors & Motors Lab

In designing the PID controller with the DC motor and IR sensor, I ran into several challenges. Originally, I had designed the Arduino code to have two interrupt pins to detect changes in both the A and B channel therefore improving the resolution of the position. Unfortunately, though probing the channels with the oscilloscope, I discovered that our DC motor had an encoder with a non-functional B channel. I could see the square waves perfectly for the A channel but not for the B channel. After spending a good amount of time testing to confirm this theory, we asked for a new motor. In the meantime, I wrote the PID controller code for the DC motor dependent only on the falling edge of the A channel thus giving 1/4th the resolution that I could have achieved using the full gray code from the encoder. Because this lower resolution controller still functioned perfectly fine in testing, I kept it as is instead of integrating the B channel back into the position reading. However, the new DC motor we received did have a functioning B channel as is shown in Fig 2.

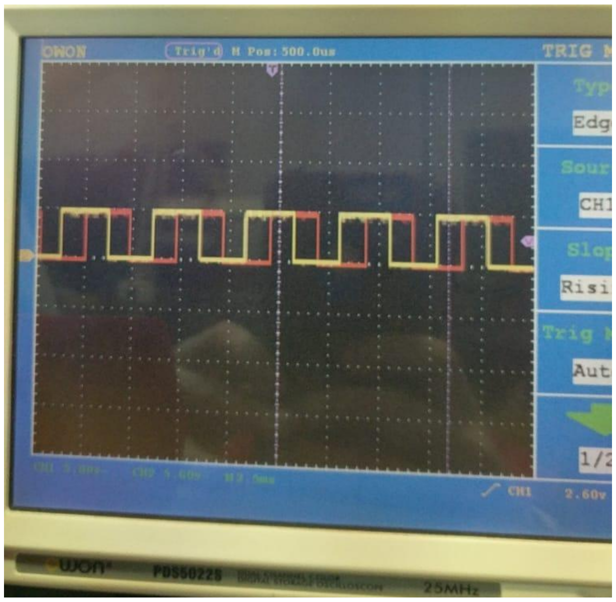

Fig 2. Oscilloscope Reading of Encoder A (Yellow) & B (Red) Channels Rotating Counterclockwise

Tuning the PID Kp, Ki, and Kd gains was also a bit of a challenge because I had started with much smaller values than needed out of practice from previously designed PID controllers. With all of the gains starting around Kp = 0.05; Kd = 0.05; and Ki = 1e-6, I got horrible behavior that was slow to respond and oscillated significantly around the desired value. The PID controller was outputting only extrema values at ±255 meaning it drove the motor at full speed one way then the other passing the desired value significantly each time. Playing with the Kp and Kd gains only seemed to vary the sluggishness of the oscillations with the gains at this scale. Eventually, I recognized that in order to improve speed I could scale up all of the gains. I ended up with Kp = 5; Kd = 10; and Ki = 0.00001. These gains created much more stable convergence, even with the reduced resolution of only readying channel A.

2.2 COBORG Project

So far, the Voice subsystem has been fairly successful with minimal challenges. I expect challenges to arise in accuracy testing in noisier environments. If these different environments cause accuracy issues that throw the system out of passing requirements, it could be very challenging to tune the system to custom audio libraries that work independently of background noise.

Additionally, it is going to be a challenge to integrate the voice_node into the main COBORG ROS structure. This is partly due to my minimal experience with ROS, but mainly due to the ever-changing structure of the main ROS state machine node. This is going to require the voice_node to be a living script requiring constant edits to fit the structure and function of the state machine node and the COBORG.

3 Teamwork

3.1 Sensors & Motor Lab

Team Member	Sensor	Motor	Motor Lab Contribution
Feng Xiang	Potentiometer	Servo Motor	-Mapped potentiometer sensor analog output to servo-motor input. -Collaborated with Yuqing to develop a control relationship between potentiometer and flex sensor with servo motor output. -Helped combine code together to get Arduino-only Arduino code state.
Jonathan Lord-Fonda	Ultrasonic	Stepper Motor	-Soldered DC motor controller board. -Wrote switch debounce code. -Wrote code structure. -Wrote program for stepper motor. -Wrote program for ultrasonic sensor.
Gerry D'Ascoli	IR	DC Motor	-Wired DC motor and motor controller to each other and Arduino. -Debugged electrical issues. -Wrote PID control of DC motor using encoder for state and IR sensor as input for desired state. -Wrote program to control stepper motor position using the ultrasonic sensor as input.
Yuqing Qin	Flex Sensor	Servo Motor	-Mapped a flex sensor output to the servo motor to control its movement. -Tested the IR sensor and implemented moving average filtering to improve the sensor outputs.
Husam Wadi	GUI		-Created a ROS publisher and subscriber node in Arduino to send/receive data -Created a serial interface between ROS and the Arduino using "rosserial" -Created a custom message class "CMU" that handled sensor data outputs -Created a URDF that controlled the motor outputs through joint_states publisher -Created a visualization in RVIZ to show the GUI motor control output -Created a RQT GUI that contained RVIZ and 5 plots that displayed all the sensor outputs in real time

3.2 COBORG Project

Team Member	Work Description for COBORG
Feng Xiang	-Developed pipeline between MoveIt and HEBI API and Coborg motors -Developed URDF and initialized moveit to move robot in RViz
Jonathan Lord-Fonda	-Wrote the initial BOM for the current system -Created initial ROS node map -Sketched out main state node -Installed Linux/ROS and completed ROS training -Tested the arm's max lift at full extension -Reviewed Jason's 3-D model of Coborg
Gerry D'Ascoli	Developed prototype of voice system: -Generated a custom implementation of pocketsphinx on the CoBorg system -Created custom libraries and recognition files to tailor the voice system to the CoBorg's specific requirements and functions.
Yuqing Qin	Worked on perception subsystem: -Implemented a YOLO ROS node -Integrated with realsense camera node
Husam Wadi	-Managed and updated project timeline to adapt to the semester load -Used scrum/agile methodologies through kanban boards to map out weekly project work -Deepened ROS interconnectivity knowledge by researching state machines in ROS and going through ROS tutorial examples.

4 Plans

4.1 COBORG Project

In the upcoming weeks, I plan to have the voice_node fully developed and ready for integration to the COBORG ROS structure. This would require tuning the accuracy to meet the COBORG requirements and building the interface between the voice_node and the COBORG main state machine node. During this time, I also plan to assist Yuqing in the Vision subsystem as many more difficulties have arisen in that area than expected. This will involve helping her develop the various ROS nodes associated with the vision subsystem including mapping the RGB image processed by the YOLO implementation to the depth map in order to better apply the YOLO output to our physical environment.

Appendix A: Arduino Code

// Encoder PID

```
#include <PID_v1.h>

//DC Motor w/ Encoder
#define enable_motor 4
#define motorpin1 5
#define motorpin2 6

int encoder_a_value = 0; // Initialize digital record of encoder_a
int count = 0;
float angle = 0;
#define encoder_a 20 // Put encoder output A pin here
#define encoder_b 21

float countpercycle = 172;

//PID
double Kp = 5;
double Kd = 10;
double Ki = 0.00001;
double Setpoint, Input, Output;
PID myPID(&Input, &Output, &Setpoint, Kp, Kd, Ki, DIRECT);

//IR Sensor
int IRpin = A5;
#define WINDOW_SIZE 5
//Averaging Filter for IR Sensor
int readings[WINDOW_SIZE];
int sum = 0;
int index = 0;

int avgFilter(int reading)
{
  sum -= readings[index];
  readings[index] = reading;
  sum += readings[index];
  index = (index+1) % WINDOW_SIZE;
  return sum/WINDOW_SIZE;
}


void setup()
```

```
{
  pinMode(encoder_a, INPUT);    // Set encoder_a to an input
  attachInterrupt(digitalPinToInterrupt(encoder_a), Increment, FALLING);  // Attach interrupt to
encoder_a
  Serial.begin(9600); // Initialize Serial Port
  Serial.println("Arduino Initialized");

  pinMode(motorpin1, OUTPUT);
  pinMode(motorpin2, OUTPUT);
  pinMode(enable_motor, OUTPUT);

  myPID.SetOutputLimits(-255,255);
  myPID.SetMode(AUTOMATIC);
}

void loop()
{
  //Calibration: Should do one cycle, adjust countpercycle variable until this is true
  /*
  while(count < countpercycle/2){
    angle = (360/countpercycle)*float(count);
    digitalWrite(motorpin1, HIGH);
    digitalWrite(motorpin2, LOW);
    Serial.println(angle);
  }

  digitalWrite(motorpin1, LOW);
  digitalWrite(motorpin2, LOW);
  delay(2000);

  while(count > -countpercycle/2){
    angle = (360/countpercycle)*float(count);
    digitalWrite(motorpin1, LOW);
    digitalWrite(motorpin2, HIGH);
    Serial.println(angle);
  }

  digitalWrite(motorpin1, LOW);
  digitalWrite(motorpin2, LOW);
  delay(2000);
  */

  //Read value from IR Sensor
  int reading = analogRead(IRpin);
  int average = avgFilter(reading);
  float voltage = (average/1024.0)*5;
  float dist = 50.6/(voltage-0.173);   // cm
  Setpoint = map(dist, 20, 150, 0, 360);

  Input = (360/countpercycle)*float(count);
  myPID.Compute();
  Serial.print(Input); Serial.print("\t"); Serial.print(Setpoint); Serial.print("\t"); Serial.println(Output);

  if(Output > 0){//CounterClockwise
    digitalWrite(motorpin1,HIGH);
    digitalWrite(motorpin2,LOW);
    analogWrite(enable_motor,Output);
  }
  else if(Output < 0){//Clockwise
    digitalWrite(motorpin1,LOW);
    digitalWrite(motorpin2,HIGH);
    analogWrite(enable_motor,-Output);
  }

}

void Increment() {
  if(digitalRead(motorpin1)==HIGH){ //CounterClockwise
    count++;
  }
  else if(digitalRead(motorpin2)==HIGH){ //Clockwise
    count--;
  }
}
```

//Ultrasonic Control of Stepper

```
#define triggerPin  13  // for ultrasonic sensor
#define echoPin 12 // for ultrasonic sensor

//Stepper Motor
// Define pin connections & motor's steps per revolution
#define dirPin 10
#define stepPin 11
const int stepsPerRevolution = 400;

int currentStep = 0;
int desiredStep = 0;

#define WINDOW_SIZE 5
//Averaging Filter for Ultrasonic Sensor
int readings[WINDOW_SIZE];
int sum = 0;
int index = 0;

int avgFilter(int reading)
{
  sum -= readings[index];
  readings[index] = reading;
  sum += readings[index];
  index = (index+1) % WINDOW_SIZE;
  return sum/WINDOW_SIZE;
}

void setup(){
  Serial.begin(9600);
  pinMode(triggerPin,OUTPUT);
  pinMode(echoPin,INPUT);

  // Stepper Motor
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
}

void loop(){
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  // Hold trigger for 10 microseconds, which is signal for sensor to measure distance.
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);

  unsigned long durationMicroSec = pulseIn(echoPin, HIGH);
  double distanceCm = durationMicroSec / 2.0 * 0.0340;  // 340m/s  0.0340cm/microsec
  if(distanceCm > 100) distanceCm = 100;
  Serial.print(distanceCm);

  desiredStep = round(map(distanceCm, 0, 100, 0, stepsPerRevolution));
  Serial.print("\t"); Serial.println(desiredStep);
  if(desiredStep > currentStep) digitalWrite(dirPin,HIGH);
  else digitalWrite(dirPin,LOW);
  for(int x = 0; x < abs(desiredStep-currentStep); x++){
     digitalWrite(stepPin, HIGH);
     delayMicroseconds(500);
     digitalWrite(stepPin, LOW);
     delayMicroseconds(500);
  }
  currentStep = desiredStep;
}
```

Appendix B: Sensors & Motors Lab Quiz

B.1 Reading a Datasheet

Refer to the ADXL335 accelerometer datasheet (https://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf) to answer the below questions.

- What is the sensor's range?

  [-3g:3g]

- What is the sensor's dynamic range?

  6g

- 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?

  Decoupling. This is achieved by C_{DC} filtering out the high frequency noise from the power supply so it does not impact the sensor and the supply voltage can remain relatively stable.

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

  $$V_{out} = 1.5V + (300\, {}^{mV}\!/_{g})a$$

- What is the largest expected nonlinearity error in g?

  ±0.3% or ±0.018g

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

  rms Noise = 948.683μg

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

  Cannot be determined from the datasheet because the datasheet gives noise performance in terms of non-zero signal frequency. This can be determined experimentally by powering the sensor, grounding the input, then measuring the passive noise output.

B.2 Signal Conditioning

- Filtering
  - Name at least two problems you might have in using a moving average filter.

    1) Since it is required at every given input, it adds computation per tick which scales to be worse as window size increases.

    2) The input value is smoothed to remove sharp changes so it would be slow to respond to actual significant changes in input.
  - Name at least two problems you might have in using a median filter.

    1) Computationally expensive, requiring selection algorithms to find the median in the windowed dataset.

    2) Peak values can never be medians, so maxes and mins are exclusively ignored.
- Op-amps: $V_{out} = V_2 + \left(\frac{R_f}{R_i}\right)(V_2 - V_1)$
  - 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 in each case: 1) which of V1 and V2 will be the input voltage and which the reference voltage; 2) the values of Rf/Ri and the reference voltage. If the calibration cannot be done with this circuit, explain why.
    - Your uncalibrated sensor has a range of -1.5 to 1.0V (-1.5V should give a 0V output and 1.0V should give a 5V output).

      1) $V_1 = V_{in}, V_2 = V_{ref}$

      2) $\left(\frac{R_f}{R_i}\right) = -2, V_{ref} = -3V$
    - Your uncalibrated sensor has a range of -2.5 to 2.5V (-2.5V should give a 0V output and 2.5V should give a 5V output).

      This calibration cannot be done because the $V_{out} = V_2 + \left(\frac{R_f}{R_i}\right)(V_2 - V_1)$ *with a mapping of [-2.5:2.5]V to [0:5]V for any variation of* V_1 *and* V_2 *as input or reference for any reference value and any* $\left(\frac{R_f}{R_i}\right)$ *slope gain.*

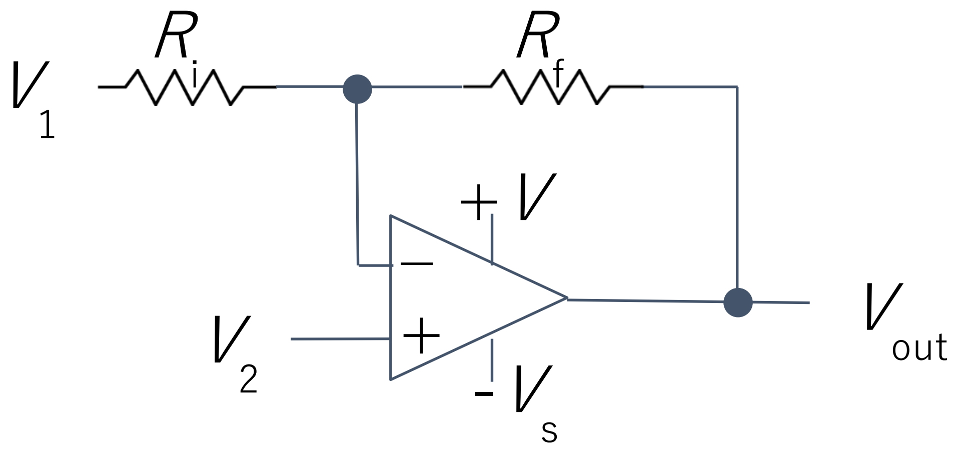

Fig. 1 Op-amp gain and offset circuit

B.3 Control

- 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.
  P: input is desired position minus current position
  *I: input is a running sum of error*timestep (error being position minus desired position)*
  D: input is previous position minus current position divided by timestep
- If the system you want to control is sluggish, which PID term(s) will you use and why?
  "P" because it increases the gain as the error increases and therefore speeds up the response time.
- 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" because it removes steady-state error over time.
- After applying the control in the previous question, if the system still has overshoot, which PID term(s) will you apply and why?
  "D" because it slows down the gain as it approaches the desired value which reduces the potential overshoot.