Sensors and Motors Lab

Individual Lab Report 1

By Clare Cui

Team B: Arcus

Clare Cui

Maitreya Naik

Angad Sidhu

Logan Wan

14 October 2016

Individual Progress

For my contribution to the Sensors and Motors laboratory, I programmed the IR proximity sensor and the stepper motor. I also collaborated in integrating the hardware system.

IR Proximity Sensor Reading and Data Processing

The IR sensor used was a Sharp 2Y0A21YK0F infrared distance measuring sensor, which is showing in Figure 1. The IR sensor sends out IR light, and, depending on the reflectivity of the surface and its distance from the sensor, returns a voltage, which is converted into analog values from 0 to 1023. To begin with, I verified that I was receiving values from the sensor by compiling the code seen in Figure 2.

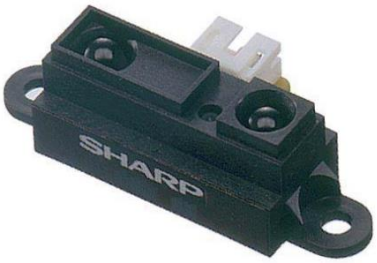


Figure 1: *Sharp IR sensor. Retrieved from* http://www.sharpsma.com/webfm_send/1489

```
int IRreading;

void setup() {
  pinMode(A4,INPUT);
  Serial.begin(9600);
}

void loop() {
  IRreading=analogRead(A4);
  Serial.print("Reading: ");
  Serial.println(IRreading);
  delay(500);
}
```

Figure 2: *Arduino code used to test that the IR sensor was functioning correctly.*

In order to get accurate conversions between the sensor and the angle, I first converted the analog reading into the voltage output. Then, I used the transfer function plot provided by the sensor data sheet – and shown in Figure 3 – in order to map the voltage to a distance using an Excel to create a polynomial function between voltage and distance. I also created a transfer function plot of my own to calibrate and tune my code to the values that our sensor was reading in. This was necessary, for example, in limiting the distances that would be used for mapping to the stepper motor, which will be discussed more thoroughly in the next section. The self-made transfer function plot can be seen in Figure 4.

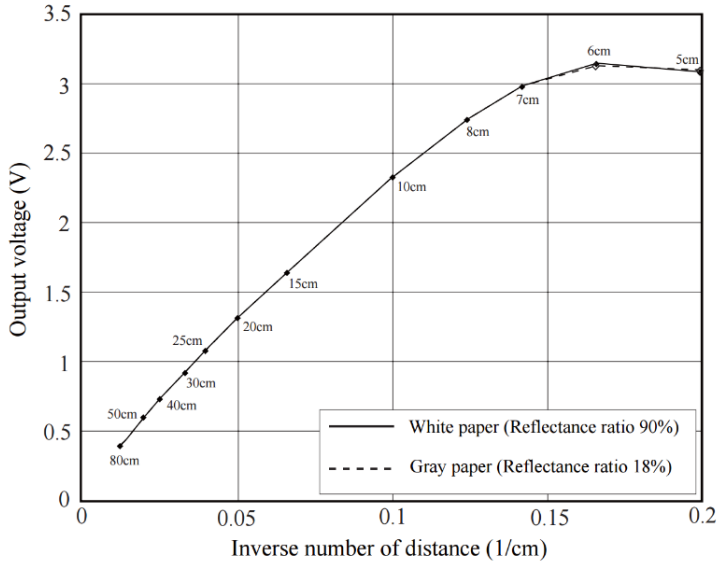


Figure 3: *Transfer function plot provided by the IR sensor data sheet at http://www.sharpsma.com/webfm_send/1489*

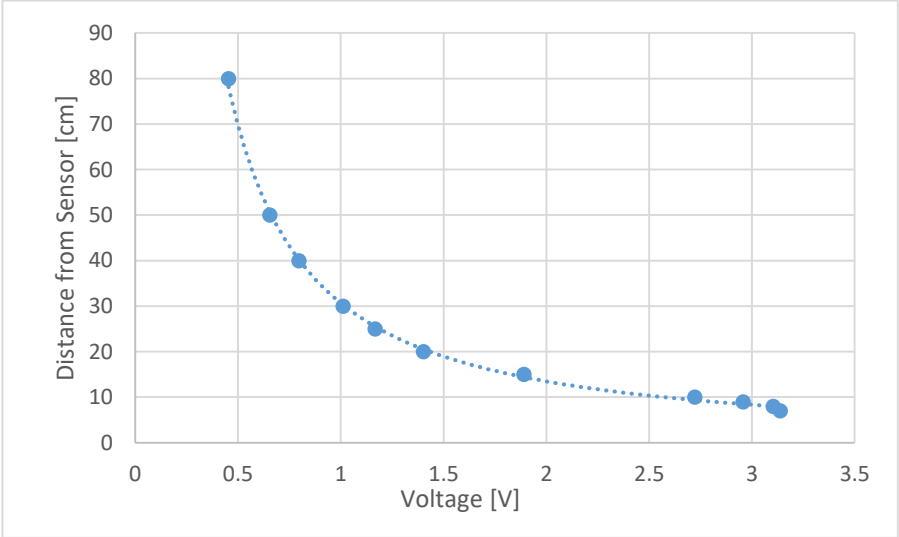
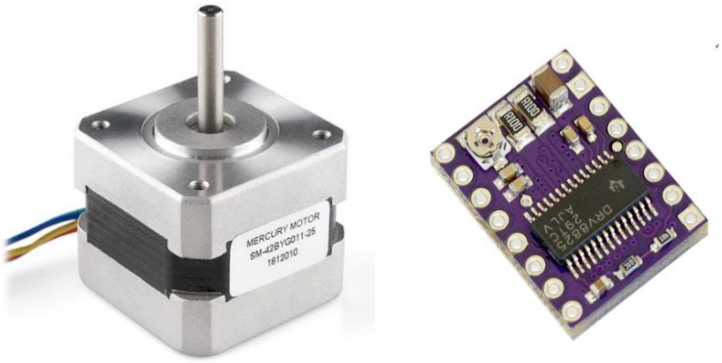


Figure 4: *Transfer function plot created from collected measurements by the team's existing sensor.*

Stepper Motor Control

The stepper motor implemented in this laboratory was a Mercury Motor SM-42BYG011-25 and was used alongside a Pololu md20b stepper motor driver, which can both be seen in Figure 5a and 5b, respectively. In order to accurately actuate the stepper motor, the distances seen by the IR sensor were mapped to angles from 0° to 360°, which were then converted into steps for the stepper motor.



(a) (b)

Figure 5: *Stepper motor (a) and stepper motor driver (b) used. Found at https://www.sparkfun.com/products/9238 and https://www.pololu.com/product/2133#lightbox-picture0J4224;main-pictures, respectively.*

The Arduino code was structured such that the stepper motor consistently tracked the distance of an object from the sensor. A target step count was determined from mapping the angle to the number of steps, and, then, a difference was taken between the target and current step count to determine both the direction and number of steps left to reach the desired angle. Because there was some error associated with the IR sensor readings, I took a running weighted average of sensed distance from the sensor so that there was less jittering in the motor given minute changes in distance. For stepping, I decided to use whole steps rather than microsteps to be able to immediately see changes in the motor angle. In order to more reliably map the sensor voltages, I empirically located consistent analog readings, which occurred between 7 cm and 71 cm. Given a distance greater than 71 cm, the motor would move back to its home position, which was located wherever the motor position was at control activation. Given a distance less than 7 cm, the motor would not move at all. The code for this can be seen as Script A1 in Appendix A. The header file that was used for software integration by Angad Sidhu can be seen as Script A2 in the same appendix. A final representation of the sensor data mapped to the angle of the stepper motor can be seen in Figure 6 below. These plots and associated GUI were created by Angad Sidhu.

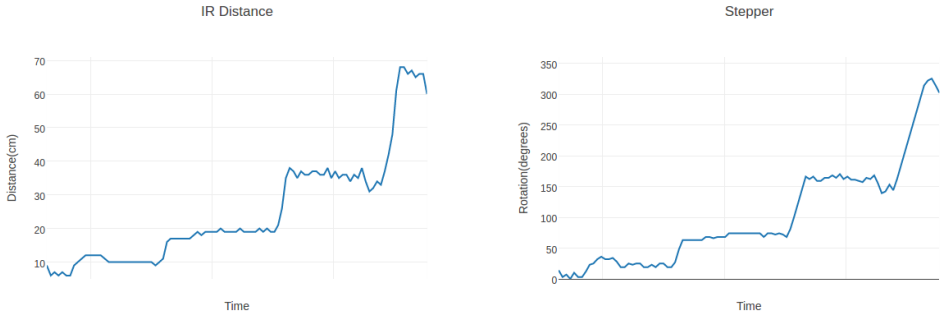


Figure 6: *IR sensor distances and resulting angles by the stepper motor shown in a graphical plot on an online GUI. GUI was created by Angad Sidhu.*

Hardware Integration

Hardware integration was taken on by me and Logan Wan. For my part, I soldered the stepper motor driver male header pins as well as set the current limit for the driver so that the stepper motor would not overheat. This required wiring up the entire driver circuit *sans* stepper motor and measuring the voltage between the V_{mot} and GND terminals to set a current that was approximately equivalent to I = 2*V, a relation given in the stepper driver specifications sheet. The current was set by adjusting a sensitive potentiometer that was mounted on the driver board itself. In addition to the driver set up, I also mounted the stepper motor, DC motor, and IR sensor on the board. Organization was done with the breadboard placement in mind such that wire crossings and lengths were minimized. A top view of the full set up can be seen in Figure 7.

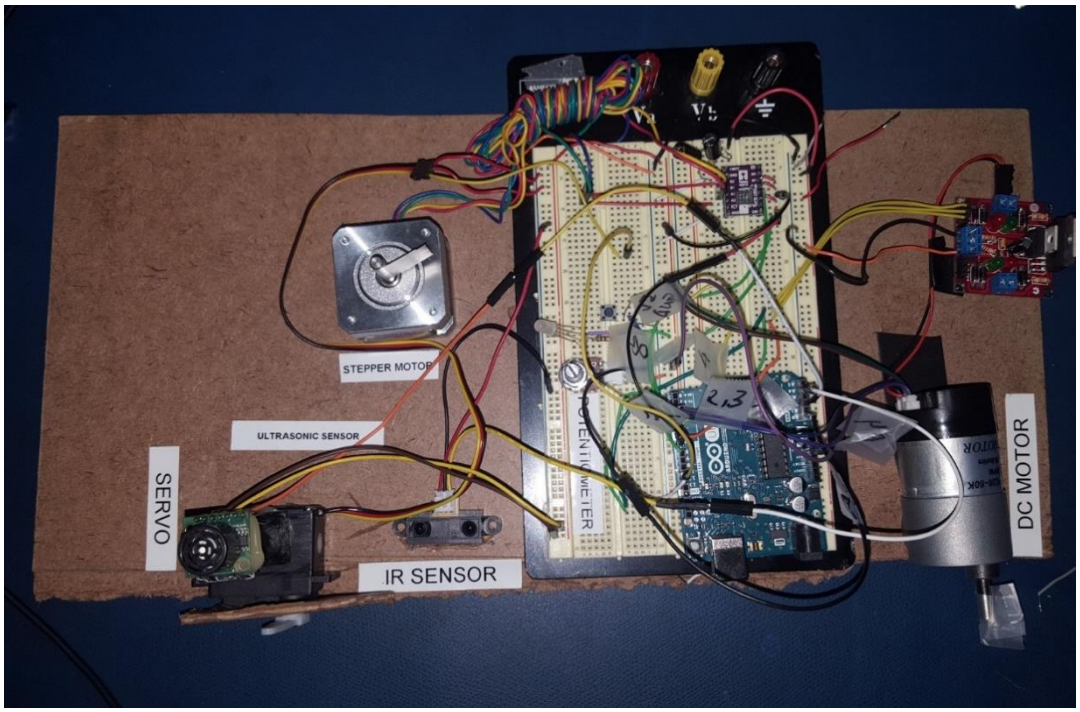


Figure 7: *Integrated sensors and motors system.*

Challenges Faced

There were several issues that were encountered in getting my subsystem up and running. First of all, the stepper driver was accidentally shorted while trying to adjust the current limits. To mitigate this, the male header strips were soldered onto the driver board upside down, such that shorting would not occur as easily and the labels for wiring were more easily visible, since they were located on the back of the board.

Another challenge was working with the software engineer (Angad) to make streamlined code that worked as quickly as possible. Originally, I had code that would execute each separate angle command to completion, meaning the code would throw out any sensor measurements that were recorded while the motor was moving and the entire system

would be blocked temporarily. By implementing a moving weighted average, a step counter that incremented or decremented one step for each loop iteration, and removing all delays in the code, this made the motor behave more intuitively, and, more importantly, did not block any background processes from the GUI and the rest of the system.

Teamwork

Clare Cui: I was responsible for the IR sensor and the stepper motor. I also integrated some of the hardware, including the stepper motor driver and mounting the IR sensor, stepper motor, and the DC motor.

Maitreya Naik: Maitreya conducted wiring and electrical integration. He took care of the PID controller for the DC motor, the potentiometer, the push button – with accompanying debouncing function – and the LED.

Angad Sidhu: Angad was responsible for the GUI and the full software integration for all of the script files from the other three team members.

Logan Wan: Logan was responsible for programming the ultrasonic sensor and the servo motor. He also did some hardware integration, including mounting the servo motor and ultrasonic sensor, which he made a custom distance indicator and 3D printed mount for, respectively.

Overall, the team worked well together in terms of dividing the tasks fairly and helping each other where needed. One aspect that could have been improved was time management, where we ran out of time to get the positioning working on the DC motor. One valuable lesson learned was to get the final code for all of the individual components to the person who was integrating all the software (Angad) as early as possible so that there was both time to modify the code appropriately and catch any other additional errors.

Current Project Progress and Future Plans

Thus far, we have bagged some data from the Lafarge quarry using the Velodyne Puck Lite and the Piksi GPS. For this, I designed a custom mount that was 3D printed on the MakerBot Replicator 2X meant to sit on top of the Velodyne. It carried the GPS antenna, the

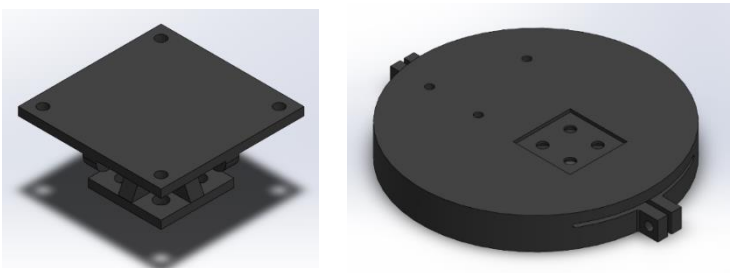


Figure 8: *Stand and base used for mounting the Piksi GPS to the Velodyne.*

Piksi itself, and the telemetry radio. The CAD model can be seen below in Figure 8, and a picture from the data bagging at the quarry can be seen in Figure 9.

Figure 9: *Bagging data at the Lafarge quarry with the Velodyne Puck Lite and Piksi GPS.*

Additionally, we have achieved RC flight of the hexrotor UAV. This was done without any sensors onboard. This was achieved by calibrating and integrating the receiver, mounting 16.8 V LiPo batteries in the undercarriage of the UAV, integrating the telemetry radio, integrating the ROS drivers, and calibrating the Pixhawk flight controller. A still from the video can be seen in Figure 10.

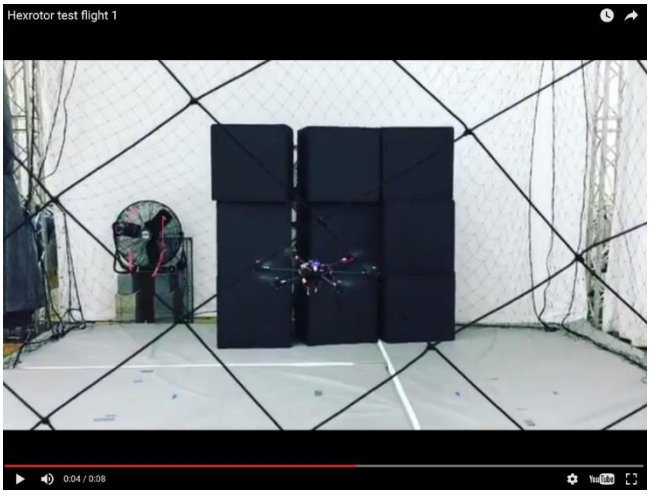


Figure 10: *Still from a video of the first RC flight of the UAV.*

In terms of the immediate future, there are several tasks at hand. Our team is working on the power distribution board initial design. We also will be working on the flight controller and onboard computer integration. The RGB camera, Velodyne, Piksi GPS, and IMU must also be brought onto the onboard computer.

My role specifically prior to the next lab demo will be in designing hardware for sensor integration onto the UAV and continuously working on CAD modeling for the entire UAV system. Since most parts were purchased off-the-shelf, I will be looking to the source websites and/or GrabCAD for pre-existing CAD models before I measure everything and CAD it myself. Logan will also be aiding me in designing mounts for the sensor integration.

Appendix A: Code

Script A1: Arduino code used for the IR rangefinder sensor and stepper motor

```
#include "Arduino.h"
#include "StepperLib.h"
float C_dist, C_volt, C_angle, C_angleDiff, C_angleOld,C_IRreading, C_IRreadingNew,
C_IRreadingOld=60.0, C_home, C_distAvg;
int C_exposedDistAvg;
int C_irPin = A4;
int C_dirPin = 8,
C_stepperPin=7,
C_mode=6,
C_steps=0, C_IRreadInt, C_i, C_IRval,C_IRval_switch,C_stepCnt=0, C_stepsHomeTarget,
C_stepsTarget, C_stepsCurr, C_angleTargetNew, C_angleTargetOld=0;
boolean rot;

void setupIR() {
  pinMode(C_irPin,INPUT);
  pinMode(C_dirPin, OUTPUT);
  pinMode(C_stepperPin, OUTPUT);
  pinMode(C_mode, OUTPUT);
  digitalWrite(C_mode, LOW);
  C_angleTargetOld = 0.0;
}

void readIR() {
  C_IRreading = analogRead(C_irPin);
  C_volt = (C_IRreading/1023)*5.0;
  C_dist = 7.8255*pow(C_volt,4)-67.253*pow(C_volt,3)+209.63*pow(C_volt,2)-
287.67*C_volt+163.78;
  C_distAvg=C_distAvg*.8+C_dist*.2;
  if (C_distAvg >= 5.612 && C_distAvg <= 71.0179){
      C_angleTargetNew = map(C_distAvg,5.6,72,0,360);
      C_exposedDistAvg = round(C_distAvg);
  }

 else if (C_distAvg > 72){

      C_angleTargetNew = 0;
    }
 else if (C_distAvg < 5){
    }
}

void writeIR(float val){
  C_stepsTarget=round(map(val,0,360,1,200));
    if (C_stepsTarget!=C_stepsCurr){
      if (C_stepsTarget-C_stepsCurr >0){ //check if this is the correct direction to
travel in
        rot = true;
        C_stepsCurr++;
      }
      else if (C_stepsTarget-C_stepsCurr < 0){ //check if this is the correct
direction to travel in
        rot = false;
        C_stepsCurr--;
      }

      digitalWrite(C_dirPin,rot);
      digitalWrite(C_stepperPin, HIGH);
      delayMicroseconds(500);
      digitalWrite(C_stepperPin, LOW);
```

```
      delayMicroseconds(500);
      C_angle = map(C_stepsCurr, 1, 200, 0, 360);
    }
}
```

Script A2: Header file for IR sensor and stepper motor

```
extern int C_exposedDistAvg, C_angleTargetNew;
extern float C_angle;

void setupIR();
void readIR();
void writeIR(float val);
```