INDIVIDUAL LAB REPORT 1

Sensor and Motor Lab

Shubham Garg

TEAM H (PHOENIX)

Teammates:

Akshit, Parv, Steve

February 21, 2019

Individual Contribution:

1. Ultrasonic sensor interfacing + Servo motor control
2. Code for push button with software debouncing to change motor state.
3. Helping teammate in writing PID code and gain tuning

Ultrasonic sensor:

Sensor Interface:

My task is to interface an LV-Maxsonar EZ series sensor with Arduino. There are three different types of interfaces possible to interface the sensor. First is by measuring the pulse width of the sensor, second by reading the analog values from the sensor and third is the serial communication-based interface. Initially, I tried using analog interface but it was prone to noise and distance measurement was very inaccurate. I couldn't use the serial interface as the baud rate is low, 9600. Also, our GUI has some internal latency which will further delay the sensor response. Pulse width based sensor data has much larger DC noise rejection. So, I decided to use PW (pulse width) based interface.

Filtering:

I measured the pulse-width using Arduino inbuilt function. For filtering, I decided to use the software filter as it is easy to implement and configurable as per the requirement. Initially, I tried averaging filter but it didn't perform well in removing outliers. Also, a large window size in the moving average filter added latency in the system. So, finally, I used a combination of Mode & median filter with the window size of 7. If the dataset has a single mode, then the filtered value is same as the mode of the dataset but if there are the multiple modes in the dataset, the median of the dataset is considered as the filtered output.

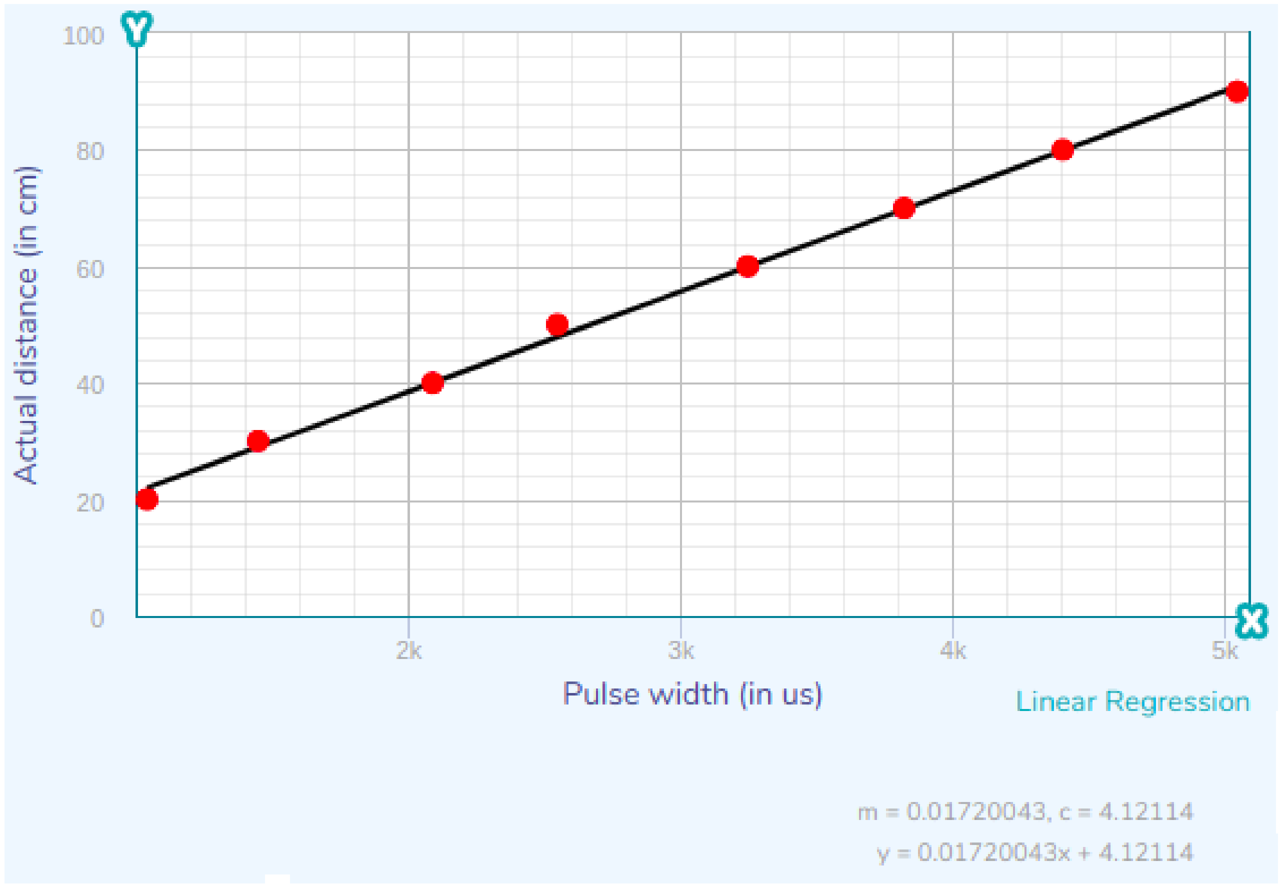
Power:

This sensor requires a voltage between 2.5 and 5.5 V and it draws around 3 mA of current. This power rating can be easily handled by the Arduino. So, I decided to power it straight from the Arduino.

Range and Transfer function:

According to the datasheet, the range of the sensor is from 6 inches to 254 inches. Our result also lies in a similar range (7 inches to 245 inches). The sensor doesn't work well below 15-18 cms.

I took multiple reading of sensor pulse width and actual distance. Below is the transfer function between the sensor pulse width in (us) v/s actual distance in meters. I used a simple least square based optimization technique to find the best fit line for the transfer function. Desired equation of the transfer function is: y = 0.0172x + 4.12114



Servo motor:

We used the HS-311 servo motor used for this assignment. Our servo motor range lies from 0-180. As this motor has an internal feedback control mechanism inbuilt in the motor, we didn't need to implement any control algorithm for position control.
The angular position of the servo motors is controlled using pulse width modulation and servo expect a pulse every 20ms. The angular position of the motor depends on the length of the pulse width. A pulse width of 1ms keeps it at 0deg and 2ms keeps it at 180deg. Varying the pulse width between 1ms and 2ms drives the servo to that particular angle. I used existing Arduino library to control the motor where the input function just needs the desired target angle and Arduino generates the required pulse width.
Further, I used the Arduino map function to map the sensor value range of 20-80cm to 0-180 servo angle.

Push button interface to change the motor state:

Internally we maintained a state machine to run different motors and sensors. There are two options available to select the motor. One is the GUI based interface and the other one is a push button.
I used software debouncing mechanism to interface it with the Arduino. Arduino just has two interrupt pins which were utilized by the DC motor encoder. So, I used a pooling-based mechanism to interface the button. Debouncing delay of around 100ms worked for us but we had to press the switch a bit longer to ensure the state is changed.

Other contributions:

We decided not to use the Arduino's PID library for controlling the DC motor. So, we wrote the PID control algorithm from the beginning. I helped him in formulating the PID feedback loop, PID input, and output parameters. Initially, the gains were pretty low and it was taking some time to reach the desired set point. I helped him in tuning the gains for the DC motor.
Like ultrasonic sensor, Force sensor also required filtering to smooth the sensor data. So, I helped Steve in developing filter algorithm for the force sensor.
I was also responsible for hardware integration especially ensuring common grounds for all the sensors and motors, keeping wire short for all the sensors to avoid any unwanted noise in the sensor data etc.

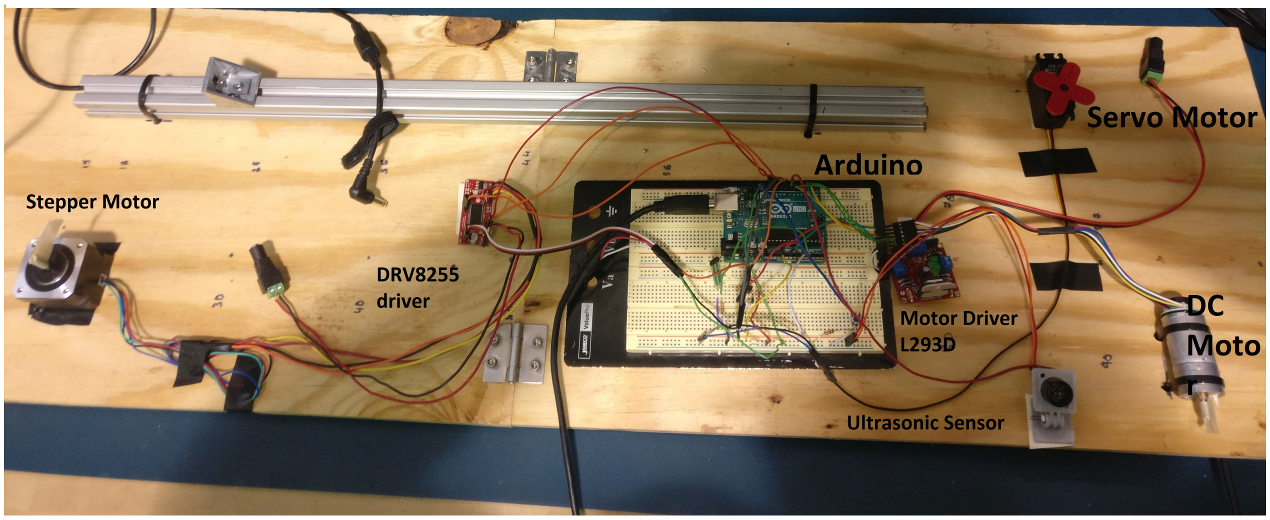
Challenges:

I faced a few difficulties in interfacing ultrasonic sensor as there were frequent spikes in the sensor values every 1 or 2 seconds. Then I tried a few filtering techniques taught in the class. I started with an averaging filter but it further increased the latency between hardware and the GUI. So, this issue was resolved by using a combination of mode & median filter. Also, initially we mounted the sensor on the slider for demonstration but we had to use long wire cables for connecting it to the Arduino. This long cable significantly introduced the noise in the sensor output. Since the ultrasonic sensor has a large beam width, it's difficult to calibrate or create a transfer function as it gets easily disturbed by the surrounding noise. So, we created a mount and slider mechanism to calibrate and develop the transfer function for the sensor.
We individually tested out sensor and motors but when it came to integration, we started facing a lot of difficulties. As everybody used their own style of programming, it was challenging to integrate all the code in a single file. We should have created the library

for individual sensor and motor which could have facilitated the sensor and motor integration.

Teamwork

Below is the complete figure of the our system. We used Ultrasonic sensor, LM35 temperature sensor and Force sensor.



Akshit: DC motor control, encoder interface, PID design and LM35 temperature sensor.
Parv: Graphical user interface and software code integration.
Steve: Flex sensor and Stepper motor.

Project progress and future plans

We have flashed the Nvidia Jetson, installed Ubuntu and other basic CUDA dependencies on it. We had to go through few iterations as display driver stopped working after installing CUDA driver. We are able to connect Zed Sensor with Jetson and get RGB images and point cloud data.
We tested the water tank with a pump but in its default state, the water pressure was not enough. In addition to that, the pump was designed to be used by keeping it in vertical, where we would need to place a tank in the horizontal position on the drone. So, without modifications, only 50-60% of the water can be pumped out. We made modification on water tank by removing the pump, sealing the hole, creating a new hole and placing the pump at that location so that it can work when placed horizontally till 100% water is deployed. We also made the nozzle small to increase the pressure so that we can shoot

water at a long distance.
We assembled the Tarot frame and were successful in doing a stable flight. We tried PX4 Alt hold mode using onboard IMU and hexacopter was going up and down by few cms. This is because barometer doesn't work well in an indoor environment, so to measure accurate distance in the z-direction, we are using Lidarlite sensor. We initially tried PWM interface but it didn't work as the latest PX4 firmware doesn't support PWM input capture. So, we used the I2C protocol for the interfacing.
We were able to setup Husky and able to drive it with manual control using rViz.
To improve the reliability of the system we are using ROS action server action which ensures that if any ROS node crashes there is a mechanism to recover from that state. Also, the user might want the ability to cancel the request during execution or get periodic feedback about how the request is progressing. We are currently integrating our few tasks like for UAVs: Land, Takeoff, for Husky: Move forward/backward, etc.
Our current plan is to use the ZED sensor for visual odometry to test the position hold. Further, the idea is to create a high-level task like such as Land, Takeoff, Localize fire & Extinguish fire.

QUIZ

Solution 1.
1. What is the sensor's range?
Ans: Minimum sensor range $\pm 3g$ and typical value is $\pm 3.6g$
2. What is the sensor's dynamic range?
Ans: Sensor's minimum dynamic range is 6g & Sensor's typical dynamic range is 7.2g.
3. 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?
Ans: Here the role of the capacitor is to remove the incoming ripple noise from the power supply. Capacitor doesn't allow the sudden change in the voltage across it. So, initially when the capacitor will be fully charged, it will constantly try to provide that voltage to the circuit and will resist the ripples from the power supply.
4. Write an equation for the sensor's transfer function.
Ans: Transfer function for this sensor is: $V_{out} = 1.5 + 0.3 * a$
where V_{out} is the electrical output from the sensor and a is the physical quantity measured which is acceleration here.
5. What is the largest expected nonlinearity error in g?
Ans: Largest expected non linearity: ± 0.3 % of the full scale. Considering the typical full scale of $\pm$ 3.6g, largest expected nonlinearity is $\pm$ 0.0108g.
6. How much noise do you expect in the X- and Y-axis sensor signals when the sensor is excited at 25 Hz?
Ans: Noise density in X & Y direction is 150 μg/$\sqrt{Hz}$. So, for 25 Hz, the RMS noise is around 0.000750 g or 750 μg.
7. How about at 0 Hz? If you can't get this from the datasheet, how would you determine it experimentally?
Ans: Noise at 0 Hz is not provided in the datasheet. But it can be determined experimentally by keeping sensor at some stationary position and taking the reading on all the three axes. Root mean square (rms) value of the reading will be the noise at the 0 Hz.

Solution 2.
Signal conditioning
1. Filtering
1.1. What problem(s) might you have in applying a moving average?
Ans:

- Moving average filter is a kind of low-pass filter which removes high frequency noise from the signal. But as the window size increases, this filter introduces latency in system which makes it unfeasible for real-time applications
- Also, this filter removes the outlier at the cost of certain information loss (because of smoothing).

1.2. What problem(s) might you have in applying a median filter?
Ans:

- Median filter requires pre-processing (sorting) of the dataset which could be a computationally expensive operation for large window size.
- Median filters can only eliminate rare/random outliers but in case if multiple such outliers occurs, median filter will fail.

2. Opamps
2.1 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:
1) which of V1 and V2 will be the input voltage and which the reference voltage;
Ans: Applying kirchhoff's law at the inverting input of the op-amp (as $I_{in} = 0$)will give following equation:

$\frac{(V_{out}-V_2)}{R_f} = \frac{V_2-V_1}{R_i}$

$$V_{out} = V_2 + \frac{R_f(V_2-V_1)}{R_i}$$

For both the cases V_{out} has to be always positive, So, only if V_1 is negative reference voltage and V_2 is input voltage, then for both the positive and negative values of V_2, V_{out} will be positive. In other cases input voltage is either always positive or negative for positive output voltage.

1. If 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). **Input should be V_2 and reference voltage should be V_1**

2. If 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) then V_1 has to positive but if V_1 is positive then the V_out will not always be positive. **So, there is no solution for this calibration.**

2) the value of the reference voltage;
Ans: For first case: If 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):

$V_2(1 + \frac{R_f}{R_i}) = V_1$

$-1.5(1 + \frac{R_f}{R_i}) = V_1$ Also,

$5 = 1 + \frac{R_f(1-V_1)}{R_i}$

Solving above two equation gave V_1 = -3V & $R_f/R_i = 1$

For Second case: If 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):

$V_2(1 + \frac{R_f}{R_i}) = V_1$

$-2.5(1 + \frac{R_f}{R_i}) = V_1$ Also,

$5 = 2.5 + \frac{R_f(2.5-V_1)}{R_i}$

So, above two equations can't be solved simultaneously. Hence there is no reference voltage for this case.

and 3) the value of Rf/Ri in each case. If the calibration can't be done with this circuit, explain why.
Ans: For first case: If 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): As solved above, **the value of $\frac{R_f}{R_i} = 1$ for this case.**

For Second case: If 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): **There exists no solution for this case as the V_1 has to positive and Resistance can't be negative. So, no solution exists for this case.**

Solution 3.
3. Control
3.1. 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.
Ans: Based on the target position, the control cmd is generated every timestep.
Below parameters are computed to fed in the PID controller. Based on the current position and desired setpoint, error is computed which is multiplied with the K_p gain.
The rate of change of error is computed by keeping track of previous error in t-1 and current error at t. Current error is substracted from previous error which is divided by the timestep (δt) and this terms is multiplied with K_d.
Finally, all the error is accumulated and multiplied with δt. This term is multiplied with the gain K_i.

u = control cmd
δt = Timestep used to control the position of the motor θ_d = Desired target position
θ_a = Actual target position/ Set point
θ_e = Error in position at any time, t
$\dot{\theta}_e$ = Rate of change of error in joint angle
$\int \theta_e$ = Accumulated error in the joint angle
$u = K_p * (\theta_d - \theta_a) + K_d * \dot{\theta}_e + K_i * \sum \theta_e$
3.2. If the system you want to control is sluggish, which PID term(s) will you use and why?
Ans: If the control is sluggish, then the P term i.e. K_p gain needs to be increased. Because K_p term controls the instantaneous error of the system and controls how quickly the system should respond if current state is away from the desired position.
3.3. 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?
Ans: If the system has steady state error, I term i.e. K_i gain needs to increased. Because K_i term accumulates the error and this adds more correction if the error persists for too long.
3.4. After applying the control in the previous question, if the system still has overshoot, which PID term(s) will you apply and why?
Ans: If the system has overshoot, then the D term i.e. K_d gain needs to be increased which introduces damping in the system. Because K_d controls the rate of change of error and limits the speed of the correction. Also, reducing K_p gain will help in reducing overshoot as it will try to make system more sluggish and reduce the overshoot.

CODE

```
#include <Servo.h>

Servo myservo;
const int pwPin = 7;
//variables needed to store values
int arraysize = 7;
int rangevalue[] = {0, 0, 0, 0, 0, 0, 0};
long pulse;
int mode;
int actual_dist;
int pos = 0;    // variable to store the servo position
int uncalib_val;

void setup()
{
  //Open up a serial connection
  Serial.begin(9600);
  //Wait for the serial connection
  delay(500);
  myservo.attach(14);
}

//Main loop where the action takes place
void loop()
{
  pinMode(pwPin, INPUT);

  for (int i = 0; i < arraysize; i++)
  {
    pulse = pulseIn(pwPin, HIGH);
    uncalib_val = pulse / 58;
    rangevalue[i] = 0.9976*uncalib_val + 4.121;
    delay(0.1);
  }
  mode = filter(rangevalue, arraysize);
  rotate_servo(int(mode));
}

//Function to print the arrays.
void printArray(int *a, int n)
{
  for (int i = 0; i < n; i++)
  {
    Serial.print(a[i], DEC);
    Serial.print(' ');
  }
  Serial.println();
}

void rotate_servo(int angle){
  myservo.write(angle);
}

void insertion_sort(int *a, int n) {
  //  *a is an array pointer function
  for (int i = 1; i < n; ++i)
  {
    int j = a[i];
    int k;
    for (k = i - 1; (k >= 0) && (j < a[k]); k--)
    {
      a[k + 1] = a[k];
    }
    a[k + 1] = j;
  }
}

//Mode function, returning the mode or median.
```

```
int filter(int *x, int n) {
  int i = 0;
  int count = 0;
  int maxCount = 0;
  int mode = 0;
  int bimodal;
  int prevCount = 0;

  while (i < (n - 1)) {
    prevCount = count;
    count = 0;

    while (x[i] == x[i + 1]) {
      count++;
      i++;
    }

    if (count > prevCount & count > maxCount) {
      mode = x[i];
      maxCount = count;
      bimodal = 0;
    }
    if (count == 0) {
      i++;
    }
    //If the dataset has 2 or more modes.
    if (count == maxCount) {
      bimodal = 1;
    }
     //Return the median if there is no mode.
    if (mode == 0 || bimodal == 1) {
      mode = x[(n / 2)];
    }
  }
  return mode;
}
```