Sensors and Motor Control Lab

Individual lab report – 01 || October 16, 2015

Team I

Pranav Maheshwari

Team Members:
Mohak Bhardwaj
Dorothy Kirlew
Richa Varma
Shivam Gautam

1. Individual Progress

I worked on the following tasks:

1. Create functional layout of microcontroller code
2. Infrared sensor integration
3. Stepper motor control

The functional layout of the program, as can be seen in Figure 1, is based on two state machines running in conjunction to decide which sensor is being used to measure a physical parameter, like rotation, force or distance. These values then control one of the actuators, servo, stepper or DC motor, depending on the state of that system.

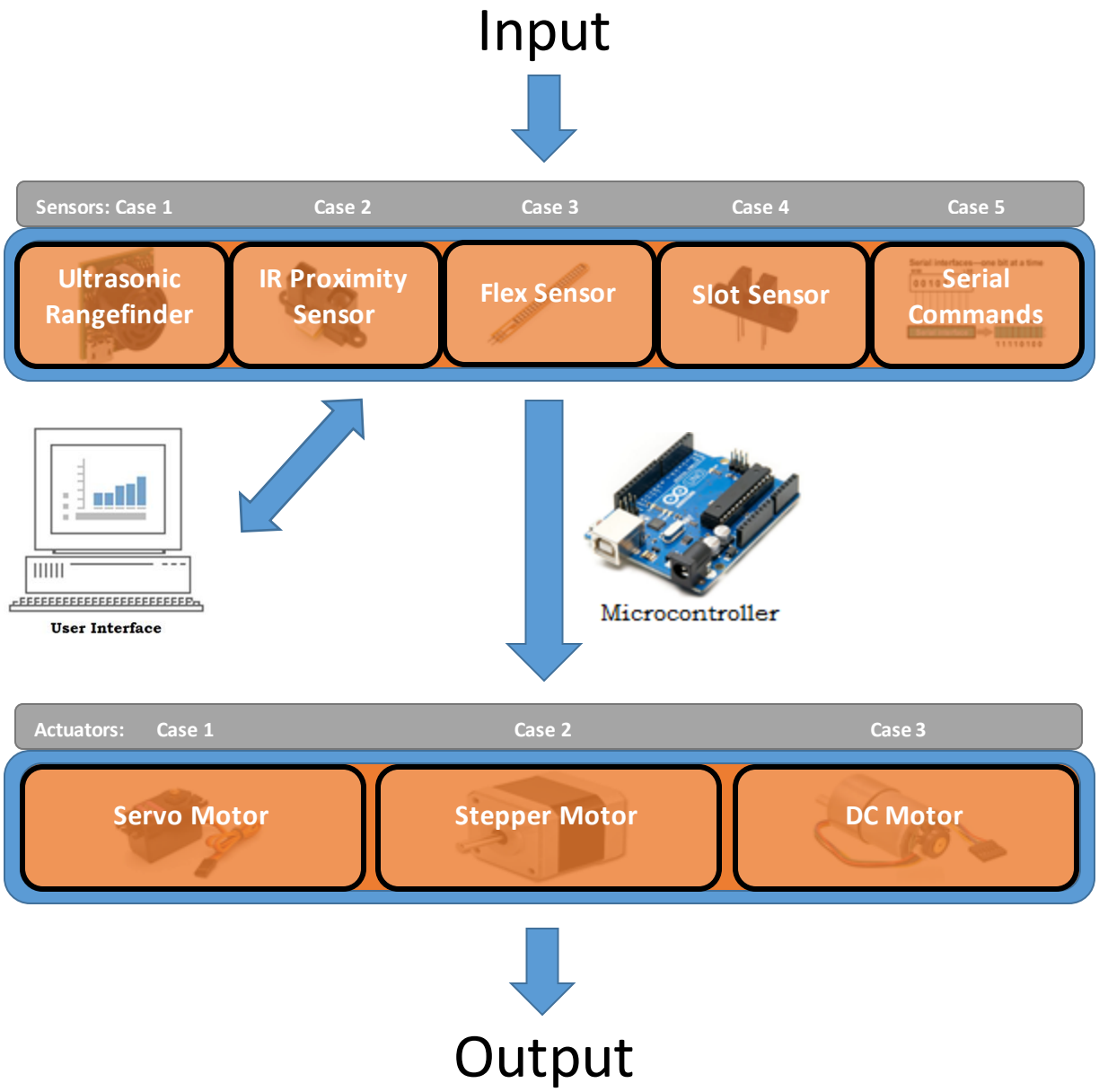

Figure 1: Functional Layout

The input can come through any of the sensors or be manually provided via GUI. These values are then mapped to an appropriate operating parameter with respect to the actuator. For example, analog readings from the force sensor are within a range of 0-1023, which is then mapped to 0-180 to a servo motor. The readings from the sensor are also constantly being communicated to the GUI via Serial communication. After writing the skeleton structure of the entire program, I proceeded to work on the SHARP IR sensor and the control code for the stepper motor.

- Infrared proximity sensor interfacing

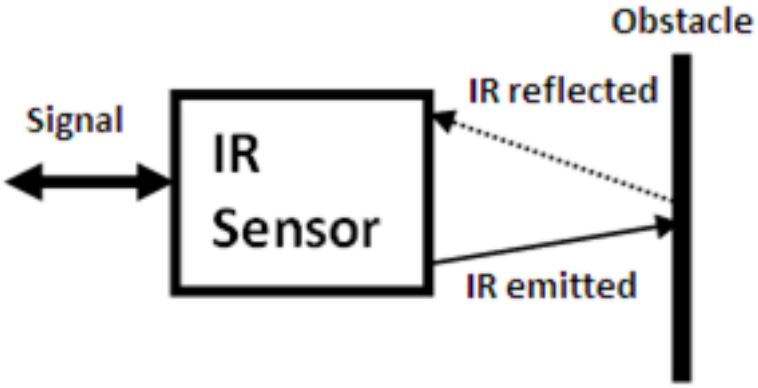

Figure 2 Working of proximity sensor

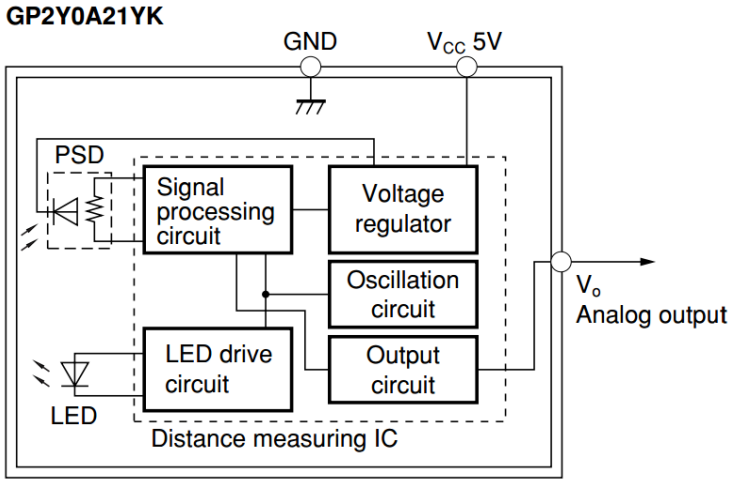

Figure 3 Internal Block Diagram of IR Proximity Sensor

Infrared proximity sensors work, as can be seen in Figure 2, by emitting IR beam and detecting the reflected signal through a position-sensible photo detector. The position of the reflected wave on the PSD helps in determining the distance.

The variant of SHARP IR sensor being used, as seen in Figure 3, has an analog output that varies from 3.1V at 10cm to 0.4V at 80cm. The distance readings being received were mapped from 10-80 to 0-1023.

- Stepper Motor control

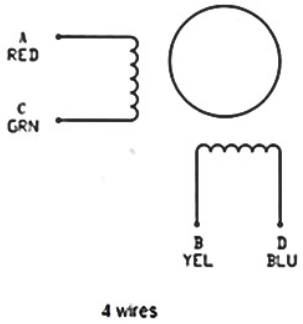

Figure 4 2 Phase Stepper Motor

The stepper motor is being driven by the SparkFun EasyDriver Integrated Circuit. It supports a resolution of a full step to a sixteenth of a step. The stepper motor provided for the task has two phases, as can be seen in Figure 4, and 200 full steps, 1.8° per step. The stepper motor requires two signals. Firstly, a constant pulse train of 50% pulse width for smooth operation to move it by certain steps. The direction is controlled through a separate signal which can be HIGH or LOW.

The number of steps required to reach a desired angle was calculated using the following equation:

$$Steps = \frac{degrees\ x\ 1\ step}{1.8°}$$

Upon receiving a signal from the sensors, the difference between present and desired position is calculated in terms of number of steps. The direction of rotation, clockwise or counter-clockwise, depends on whether the difference between present and desired value is positive or negative.

2. Challenges

Once I gave proper structure to the entire code, everyone knew where there piece of code is supposed to go, which eased the task of integration. While working on the IR sensor, after reading the datasheet and performing some initial testing, I discovered that it tends to randomly give out garbage values as output which disturbs the working of the system. To account for this, I decided to create a buffer of 30 values, filter out the ones deviating more than +5% from neighbouring values and take a mean of the rest. The IR output became much more consistent and accurate after implementing this approach. Another challenge was to setup the entire rig. Due to the variety of the sensors and actuators being used, it was important to arrange everything in a coherent manner, so that it is easy to debug and demonstrate.

3. Teamwork

Team Mate	Role
Mohak Bhardwaj	Worked on the development of the GUI. Collaboration: Worked together on creating the protocol for communication between Arduino and GUI.
Dorothy Kirlew	Worked on the development of the GUI and the ultrasonic sensor. Collaboration: Worked together on establishing robust communication between Arduino and the GUI.
Shivam Gautam	Worked on the PID control of the DC motor and the flex sensor. Collaboration: Worked together on the integration of the PID code within the main program.
Richa Varma	Worked on the slot sensor, servo motor and creating the test rig. Collaboration: Worked together on testing individual sensors, actuators and the complete final setup.

4. Plans

Over the next couple of weeks, we plan on having the mobile test platform and rigging it up with camera and sensors. Once that is done, I'll start working on fusing proximity sensor data with computer vision to detect and localize obstacles in robot's path. I'll also be working on creating a ROS based architecture which supports multiple masters, multiple robot platforms in our case. This will help us in providing communication channel over which the robots can collaborate.

5. Code

Microcontroller Code:

```
#include<SharpIR.h>
#include<PID_v1.h>
#include<Servo.h>
#include<Encoder.h>
#define m1 5 //motor pin 1 (Grey Wire from Motor Board)
#define m2 6 //motor pin 2 (Yellow Wire)

//Declare you sensor and actuator related constants
Servo myservo;
int ir = A0;
int slot = A1;
int flex = A2;
int ultra = 7;
```

```
SharpIR sharp(ir, 30, 95, 1080);
int stepper_signal = 13;
int stepper_direction = 12;
int serv;
Encoder myEnc(2, 3);
void velocityControl( int m_speed, int m_direction);

int flag = 0, prevState=100;

//PID stuff
double Kpv = 2, Kiv = 0.5, Kdv = 0.03;
double Kp = 3.4, Ki = 0.22, Kd = 0.14;
double Setpoint=0, Input=0, Output=0;
double Setpoint_v = 0, Input_v = 0, Output_v = 0;
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
PID myPID_v(&Input_v, &Output_v, &Setpoint_v, Kpv, Kiv, Kdv, DIRECT);


//Declare the pins here. 2,3,5,6,9,12 and 13 are occupied by DC, Servo and
Stepper
//int control = 11;
int button = A4;

//Declare your sensor related constants here. Use sensor_val for your senso
r.
int sensor_val = 0;
int stepper_last = 0;
int sensor_control = -1;
int actuator_control = -1;
long debounceDelay = 50;
long lastDebounceTime = 0;
boolean state = LOW;
boolean last = LOW;
int oldactuator;
int val;
int prev_sensor_val = 0;
long previousMillis = 0;
long previousMillis2 = 0;
long interval = 100;
unsigned long currentMillis;
int desiredAngle = 0, currentPosition = 0, desiredPosition = 0, error = 0;
float setpoint = 0;
int m_speed = 0, m_direction = 1;
double desiredSpeed = 0, currentSpeed = 0;
int ultra_fix = 0;
int current = LOW;

//Declare your serial communication related constants here.
String incoming = "";
char test;
String temp;

//Setting up Serial and pin modes.
void setup()
{
  Serial.begin(9600);
  pinMode(m1, OUTPUT);
  pinMode(m2, OUTPUT);
  pinMode(slot, INPUT);
  pinMode(flex, INPUT);
  pinMode(stepper_signal, OUTPUT);
```

```
  pinMode(stepper_direction, OUTPUT);
  pinMode(ir, INPUT);
  pinMode(button, INPUT);
  //pinMode(control, OUTPUT);
  pinMode(ultra, INPUT);
  myservo.attach(9);
  myPID.SetMode(AUTOMATIC);
  currentPosition = myEnc.read();
  myPID.SetOutputLimits(-255, 255);
}

//Serial Communication: Two formats of inputs can be processed. C-A-
XXXX where C is control of Actuator A {0,1,2} by Value XXXX {0-1023}.
// The other format is S-A-
T where S is sense transducer T {0,1,2,3} to control Actuator A {0,1,2}.
void comm()
{
  if (Serial.available() > 0)
  {
    String incoming = "";
    while (Serial.available() > 0)
    {
      test   = char(Serial.read());
      incoming  += test;
      delay(100);
    }
    temp = incoming.substring(0, 1);
    if (temp == "C")
    {
      actuator_control = incoming.substring(1, 2).toInt();
      sensor_control = 4;
      sensor_val = incoming.substring(2, incoming.length()).toInt();
      setpoint = map(sensor_val, 0, 1023, 0, 720);
      currentPosition = myEnc.read();
      Setpoint = currentPosition + setpoint;
      Setpoint_v = sensor_val * 0.0008211;
     // curState=prevState+1;
    }
    else if (temp == "S")
    {
      actuator_control = incoming.substring(1, 2).toInt();
      sensor_control = incoming.substring(2, 3).toInt();
    }
  }
}

//One button to control them all!
void debounce(int value_button){
  if (value_button>1000)
  {
    current = HIGH;
  }
  else
  {
    current = LOW;
  }
  if (current == last)
  {
    lastDebounceTime = millis();
  }
```

```
  else if (current != last && (millis() - lastDebounceTime) >
debounceDelay)
  {
    last = current;
    state = !state;
    if (state == HIGH)
    {
      Serial.print("O,O,O,");
    }
    else
    {
      Serial.print("X,X,X,");
    }
  }
}

void velocityControl( int m_speed, int m_direction)
{
  if (m_direction == -1)
  {
    analogWrite(m1, m_speed);
    digitalWrite(m2, LOW);
  }
  else if (m_direction == +1)
  {
    analogWrite(m2, m_speed);
    digitalWrite(m1, LOW);
  }
  else
  {
    digitalWrite(m1, LOW);
    digitalWrite(m2, LOW);
  }
}

//Pranav
void sharp_sensor()
{
  sensor_val = sharp.distance();
  if ((sensor_val > 5) && (sensor_val < 81))
  {
    sensor_val = (sensor_val - 6) * 13;
  }
  else
  {
    sensor_val = 0;
  }
}

//Richa
void slot_sensor()
{
  sensor_val = 0;
  if (analogRead(slot) > 1000)
  {
    sensor_val = 1023;
  }

}

//Dorothy
```

```
void ultrasonic_sensor()
{
  // Get base reading.  Subtract 818 to get values in correct range.
  // Multiply by 0.017 to change to inches.
  // Multiply by 36.54 to convert from inches to 0-1023 range
  sensor_val = 0;

  int lastRead = ((pulseIn(ultra, HIGH) - 818) * 0.017) * 36.54;
  int temp;
  // sum 10 good readings that will be averaged to filter out noise
  for (int i = 0; i < 10; i++)
  {
    temp = ((pulseIn(ultra, HIGH) - 818) * 0.017) * 36.54;
    if (temp > 1100)
    {
      sensor_val += 1023;
    }
    // if out of range or too different from last reading, get a new
reading
    else if (temp < 0 || abs(temp - lastRead) > 50)
    {
      i--;
    }
    // if reading is acceptable, sum it
    else
    {
      sensor_val += temp;
    }
    lastRead = temp;
  }
  // if average is too high, just return 1023.  otherwise return average.
  if ((sensor_val / 10) > 1023)
  {
    sensor_val = 1023;
  }
  else
  {
    sensor_val = sensor_val / 10;
  }
}

//Shivam
void flex_sensor()
{
  int frc = analogRead(flex);
  float voltage = frc * (5.0 / 1023.0);
  float frcRes = ((5 - voltage) * 10000) / voltage; //frc resistance=
((Vcc-Va)*R)/Va
  float force = 0;
  if (frcRes > 80000)
  {
    force = 0;
  }
  else
  {
    force = 1 + ((-frcRes + 80000) / 77500);
  }
  int force_map = force * 100;
  sensor_val = map(force_map, 0, 200, 0, 1023);

}
```

```
void serial_handler(String X)
{
   setpoint = map(sensor_val, 0, 1023, 0, 720);
 if (abs(prevState-setpoint)>40)
    {
  currentPosition = myEnc.read();
  Setpoint = currentPosition + setpoint;
  Setpoint_v = sensor_val * 0.0008211;
  flag = 1;
  prevState=setpoint;
    }

  if (currentMillis - previousMillis > interval)
  {
    previousMillis = currentMillis;
    Serial.print(X);
    Serial.print(sensor_val);
    Serial.print(",");
  }
}

void loop()
{
  currentMillis = millis();
  comm();
  debounce(analogRead(button));
  switch (sensor_control)
  {
    case 0:
      slot_sensor();
      serial_handler("S");
      break;
    case 1:
      ultrasonic_sensor();
      serial_handler("U");
      break;
    case 2:
      sharp_sensor();
      serial_handler("I");
      break;
    case 3:
      flex_sensor();
      serial_handler("T");
      break;
    default:
      //check for some stuff, example sensor_val value, if received through
Serial.
      break;
  }

  switch (actuator_control)
  {
    case 0:
      serv = map(sensor_val, 0, 1023, 0, 180);
      myservo.write(serv);
      delay(20);
      break;
    case 1:
      if (prev_sensor_val != sensor_val)
      {
```

```
        val = (sensor_val - prev_sensor_val) / 5.12; //Sensor_val to steps
conversion with difference between present and desired state
        if (val > 0)
        {
          digitalWrite(stepper_direction, HIGH);
          while (val > 0)
          {
            val--;
            digitalWrite(stepper_signal, HIGH);
            delay(5);
            digitalWrite(stepper_signal, LOW);
            delay(5);
          }
        }
        if (val < 0)
        {
          digitalWrite(stepper_direction, LOW);
          while (val < 0)
          {
            val++;
            digitalWrite(stepper_signal, HIGH);
            delay(5);
            digitalWrite(stepper_signal, LOW);
            delay(5);
          }
        }
        prev_sensor_val = sensor_val;
      }
      break;

    case 2:
      for (int i = 0; i < 300; i++)
      {
        Input = myEnc.read();
        myPID.Compute();
        m_speed = abs(Output);
        m_direction = Output > 0 ? 1 : -1;
        velocityControl( m_speed, m_direction);
        delay(5);
      }
      velocityControl( 0, m_direction);
      break;

    case 3:
      float initialCount = myEnc.read(), finalCount = 0;
      delay(50);
      finalCount = myEnc.read();
      currentSpeed = (finalCount - initialCount) / 50;
      Input_v = currentSpeed;
      myPID_v.Compute();
      m_direction = 1;
      float spMap = (Setpoint_v * 255) / 0.84 ;
      m_speed = Output_v + spMap;
      velocityControl( m_speed,  m_direction);
      break;
  }
  oldactuator = actuator_control;
}
```