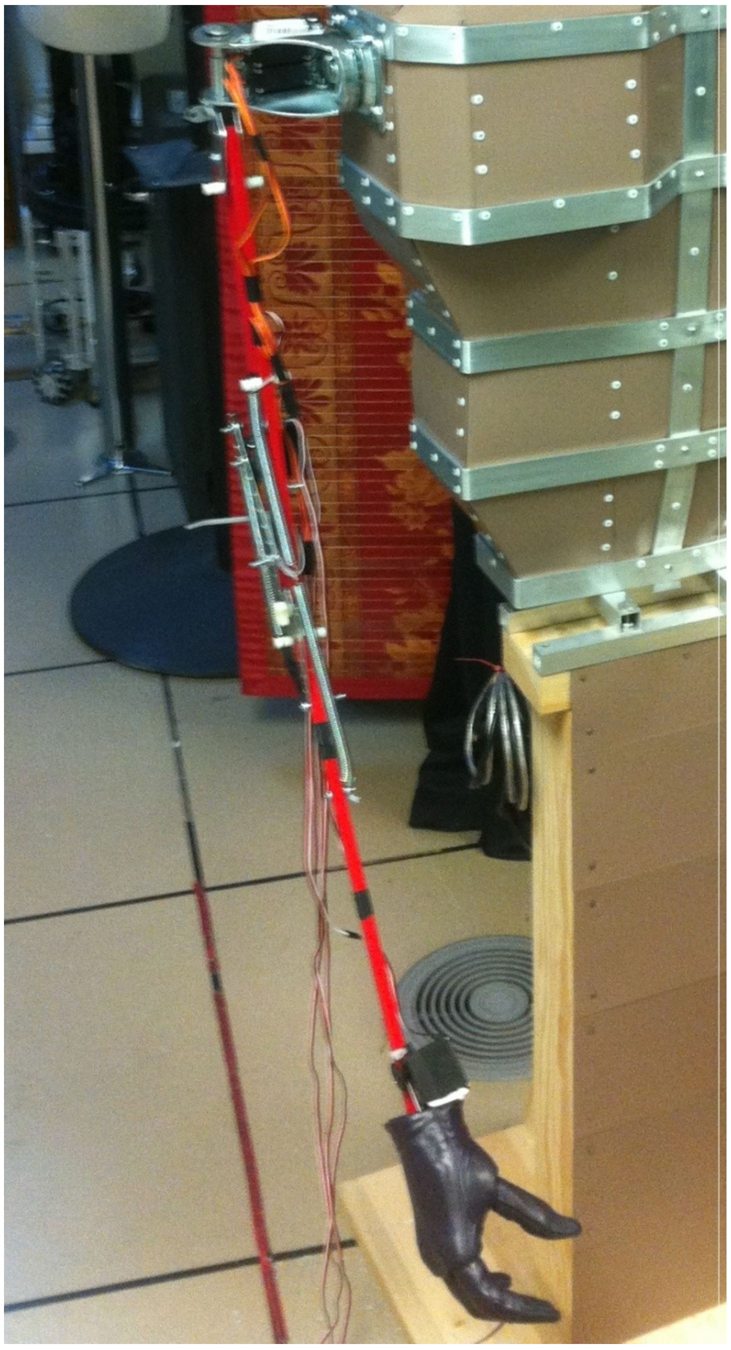
Embedded Robotics **ECE 478/578 – Fall 2012** **Final Report**

The Handshaking Robot Arm

Project partners:

Michael Engstrom Jason Peterson



Overview

The original concept for this project was a robot arm repurposed from a lamp base that would use sensors to detect the presence of an approaching hand and move to shake hands. Servos would be used to move the segments of the arm so that the robot hand on the arm would move to where the detected human hand is located. Control logic would be handled by a programmed control board.

We used an existing robot hand that had one degree of freedom (all fingers open and close) and designed the control board, sensor array and servo configuration. The basic operation of the Robot Arm design would be for interaction with a user who would approach the arm. We would need a working handshaking arm to have a successful project outcome.

Basic Project Description

- Arm repurposed from jointed lamp
- Servos mounted on arm to move with 4 degrees of freedom
  - Shoulder up/down
  - Shoulder left/right
  - Elbow up/down
  - Hand open/close
    - Robot hand consisting of five fingers moved together servo motor control
- Orangutan control circuit with logic for enabling arm and hand movement
- Distance sensors for detecting approaching object
  - Ultrasonic sensor array
    - Detect quadrant of approaching object
    - Calculate distance of approaching object
  - Optical sensor
    - Mounted on hand
    - Shorter range
    - Detect when handshake occurs

Project Planning and Implementation

We collected some supplies from the Robotics Lab, such as servo motors and metal brackets for the hand. Our choice of control board, after some web searching, was the Orangutan SVP 1284 board. This board has eight onboard hardware servo controllers, two motor controllers, three serial interfaces (on USB and two UART) and button inputs. Outputs include LEDs and an attached backlit 2x14 character LCD, as well as multiple programmable IOs. The Orangutan can be powered and programmed via USB, but for servo use a battery pack power supply was necessary. We purchased high-powered servos to control the shoulder and elbow; the hand had a servo already attached. Four Parallax ultrasonic sensors (3 meter range) were purchased for our sensor array, and one infrared optical sensor was used for close-up object detection.

The Orangutan boards can be purchased at Pololu's website for about $100, and downloads are available with many examples for the boards. Someone with no previous knowledge of robot controllers (but with some C++ programming experience) can quickly implement, compile and flash example designs to the Orangutan board and experiment with modifying the behavior.

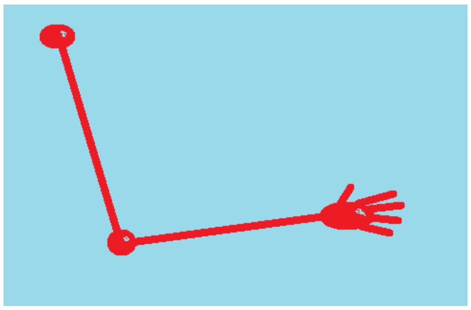
We have found that a bottom-up design can be useful for smaller projects. This project might be considered medium-sized and it can be difficult to combine smaller projects. However, since we had the opportunity to split the tasks up into HW1 and HW2 we kept the bottom-up methodology. Homework 1 would implement the inverse kinematics, Homework 2 would implement the search algorithm, and the Final Project would combine Homework 3 (fuzzy logic) to determine Handshaking Arm location based on sensor inputs.

Project Design

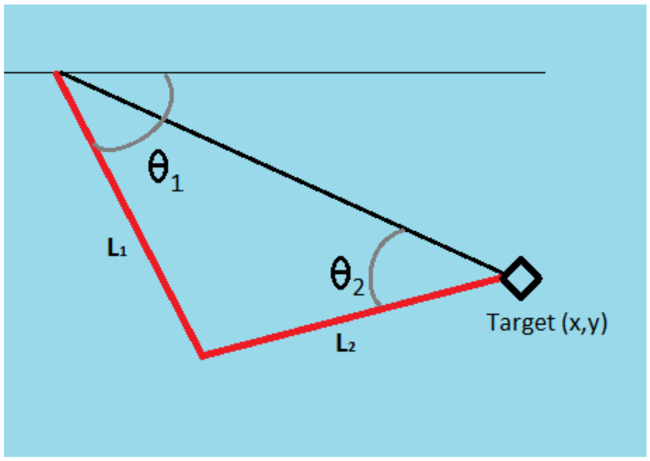
This design is for a robot arm with 4 degrees of freedom, two for the shoulder, one for the elbow, and one for the hand. We used the Orangutan development board from Pololu for the hardware implementation. The inputs are distance sensors for quadrant object detection and an optical sensor for detection of a user hand in the proximity of the robot hand. Outputs are control signals to be used for servos connected to the robot arm, moving the hand into the correct quadrant of the detected user hand. The control board has a 2-line by 14 character display and the sensor quadrant values are displayed. The current quadrant is shown also with a LED array. Once the user hand is detected, the robot hand closes for the handshake.

Inverse Kinematics Implementation

With a 2 DOF (shoulder, elbow) model we are able to bypass some of the trickier math involving transforms and translations and use the Law of Cosines with a simple triangle to find the angles needed to place the hand at the target. Here is an example of the Arm:



And here is the geometric representation with a desired target:



From the Law of Cosines we have:

$$c^2 = a^2 + b^2 - 2ab\cos(c)$$

Inserting our values and solving for the desired angles gives us:

$$\theta_2 = \arccos\left[\frac{x^2 + y^2 - L_1{}^2 - L_2{}^2}{2L_1L_2}\right]$$

And

$$\theta_1 = \arcsin\left[\frac{L_2\sin(\theta_2)}{\sqrt{x^2+y^2}}\right] + \arctan 2\left(\frac{x}{y}\right)$$

Using the above angle formulas and some input values, a simple c++ program was written to calculate the output angles needed to place the hand at the target. Below is an example of the program running. Note that the angle calculations are converted from radians to degrees by multiplying by ($\frac{180}{\pi}$¿. When the function is integrated into our control program the degrees will be converted to the pulse width necessary to move the servos to the desired angles.

There are some problems with this method for our use, but they are resolved with logic. The θ_1 value is not unique; the solution can be in the first quadrant or fourth quadrant and be a viable solution. Therefore some solutions are not natural solutions for humanoid movement (most people are unable to bend their elbow two directions). Some logic is introduced to not allow certain combinations of angles.

Also, if the target is out of range a decision needs to be made to: not move the arm until it is in range; have the arm extended toward the target as it approaches. A more human-like method is to wait until the target (human hand) is within range before moving.

Sensor Quadrant Logic

	Quadrant 1	**Quadrant 2**	**Quadrant 3**	**Quadrant 4**
Shoulder Up/ Down	Up	Up	Up	Up
Shoulder Left/Right	Left 1/4	Left 1/2	Left 1/4	Left 1/2
Elbow Up/Down	1/2 Up	1/2 Up	1/4 Up	1/4 Up
Hand Open/Close	Close when hand detected	Close when hand detected	Close when hand detected	Close when hand detected

Handshaking Robot Arm Counter Pseudocode

- Sensor data board
  - Get Ultrasonic Sensor data
    - Calculate detected object quadrant
    - Calculate distance
  - Get Optical Sensor data
  - Send data to serial data bus
- Control data board
  - Input serial data from sensor control board
  - Parse serial sensor data
    - Byte 1
      - Optical sensor

            - 0 = no object in proximity to robot hand
        - Byte 2
            - Quadrant of detected user hand
        - Bytes 3 and 4
            - Distance of detected hand
- Interaction Model
    - If nothing detected by sensor array
        - Keep robot arm lowered
    - If object detected
        - Send quadrant and distance information
    - Once object in range (50 – 60 cm)
        - Send pulse data to servos for arm position
        - If object detected on hand-mounted optical sensor (0-10 cm)
            - Wait 1 second
            - Close hand
            - Wait until optical sensor indicates hand is no longer present
    - Wait for next object detection

Arm Construction

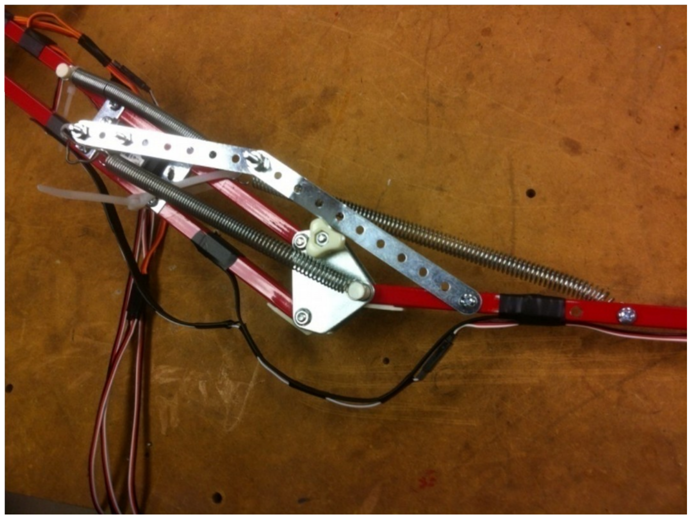


Figure 1 - Elbow Open

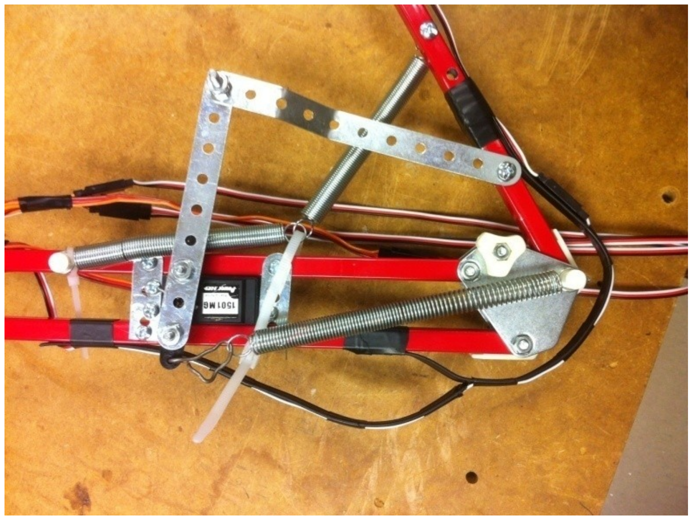


Figure 2 - Elbow Closed

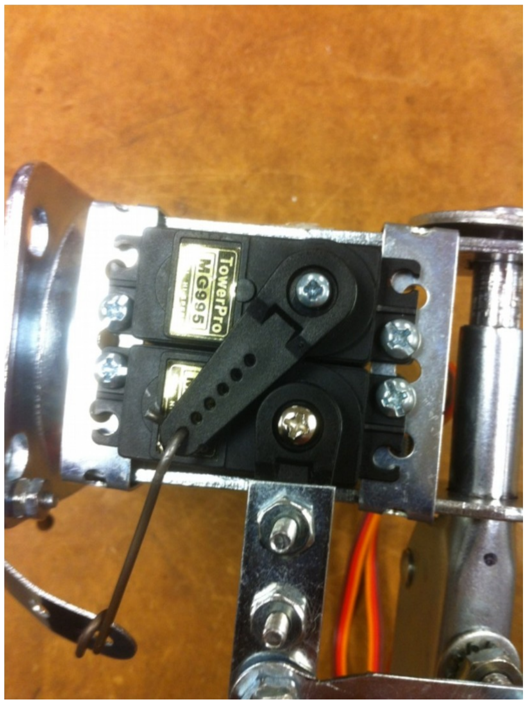


Figure 3 - Shoulder Down

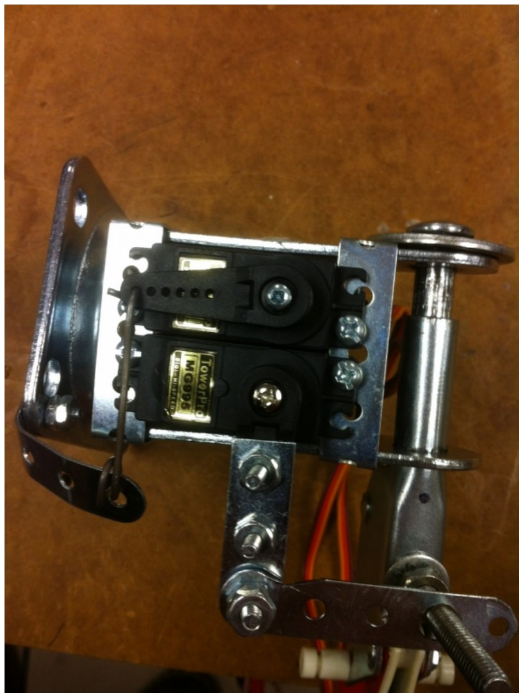


Figure 4 - Shoulder Up

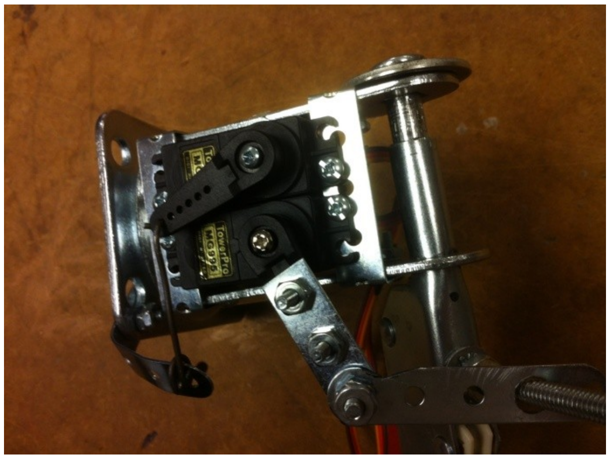


Figure 5 - Shoulder Lef

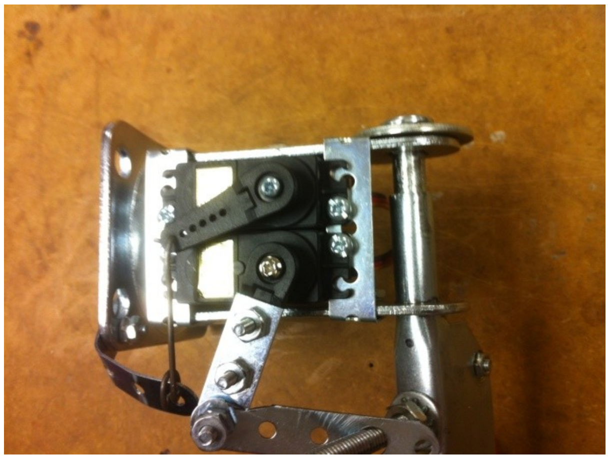


Figure 6 - Shoulder Right

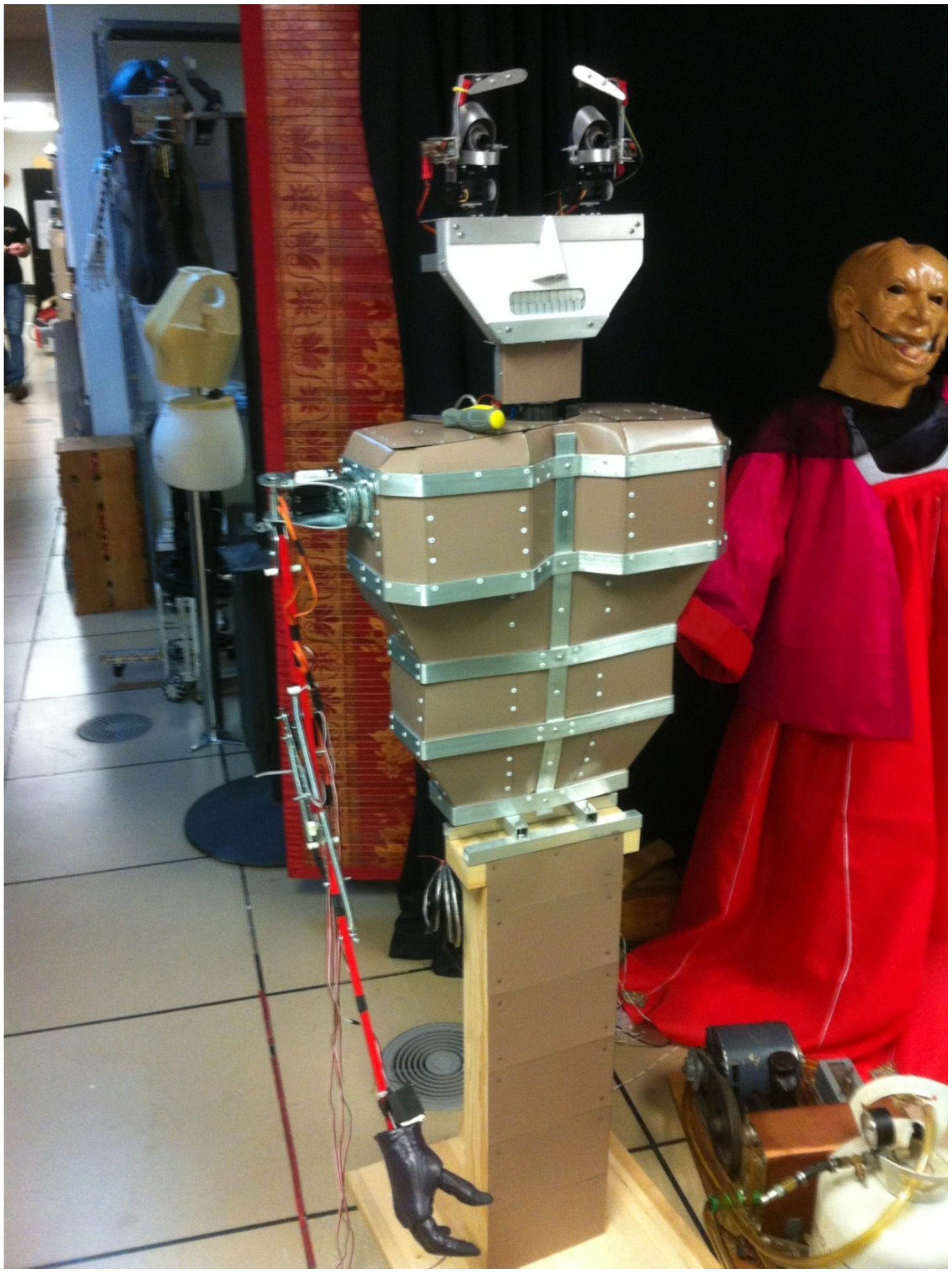


Figure 7 - Arm Mounted on 'Dim' Torso

Sensor Configuration

Orangutan C++ Control Code

```c
#include <pololu/orangutan.h>
#include <string.h>

/*
 * The Handshaking Robot Hand: for the Orangutan SVP and Orangutan X2 controllers.
 *
 * This program listens for serial bytes transmitted via USB to the controller's
 * virtual COM port (USB_COMM).  Whenever it receives a byte, it performs a
 * custom action.  The servo positions are set for specific quadrant placement of
 * the robot hand to meet the user hand for handshaking.
 *
 * http://www.pololu.com/docs/0J20
 * http://www.pololu.com
 * http://forum.pololu.com
 */
 /*
 * To use the SERVOs, you must connect the correct AVR I/O pins to their
 * corresponding servo demultiplexer output-selection pins.
 *   - Connect PB3 to SA.
 *   - Connect PB4 to SB.
 */

// This array specifies the correspondence between I/O pins and DEMUX
// output-selection pins.  This demo uses three pins, utilized to control the
// four degrees of freedom

const unsigned char demuxPins[] = {IO_B3, IO_B4}; // four servos

static unsigned char init_speed = 150;
static unsigned int neutral_servo_pos = 1300;
static unsigned int shoulder_up = 300;
static unsigned int shoulder_dn = 1500;
static unsigned int shoulder = 1800;
static unsigned int shoulder_rot_lt = 1600;
static unsigned int shoulder_rot_rt = 600;
static unsigned int shoulder_rot = 1000;
static unsigned int elbow_up = 2450;
static unsigned int elbow_dn = 1500;
static unsigned int elbow = 1800;
static unsigned int hand_open = 2000;
static unsigned int hand_closed = 1300;
static unsigned int hand = 1700;

// receive_buffer: A ring buffer that we will use to receive bytes on USB_COMM.
// The OrangutanSerial library will put received bytes in to
// the buffer starting at the beginning (receiveBuffer[0]).
// After the buffer has been filled, the library will automatically
// start over at the beginning.
char receive_buffer[32];

// receive_buffer_position: This variable will keep track of which bytes in the receive buffer
// we have already processed.  It is the offset (0-31) of the next byte
// in the buffer to process.
unsigned char receive_buffer_position = 0;

// send_buffer: A buffer for sending bytes on USB_COMM.
char send_buffer[32];
```

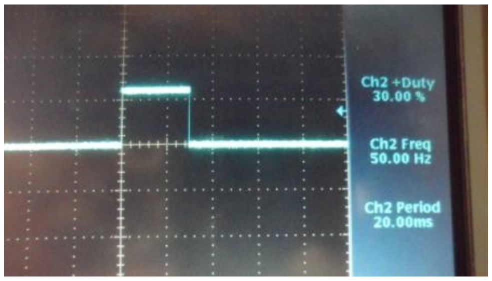
```
// sensor_buffer: A buffer for holding sensor bytes received on USB_COMM.
char sensor_buffer[5];

int byte_counter = 0;

// wait_for_sending_to_finish:  Waits for the bytes in the send buffer to
// finish transmitting on USB_COMM.  We must call this before modifying
// send_buffer or trying to send more bytes, because otherwise we could
// corrupt an existing transmission.
void wait_for_sending_to_finish()
{
      while(!serial_send_buffer_empty(UART1))
            serial_check();          // USB_COMM port is always in SERIAL_CHECK mode
}

// process_received_byte: Responds to a byte that has been received on
// USB_COMM.  If you are writing your own serial program, you can
// replace all the code in this function with your own custom behaviors.
void process_received_byte(char byte)
{
      clear();           // clear LCD

      switch(byte)
      {
            // If the character 'X' or 'x' is received, increment counter.
            case 'x':
            case 'X':
                  byte_counter += 1;
                  break;
            // After third ‘x’ byte, populate char array. Only input numbers.
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                  if(byte_counter >= 7)
                  {
                        break;
                  }
                  if(byte_counter == 3)
                  {
                        sensor_buffer[0] = byte; // optical hand detection byte
                        byte_counter += 1;
                        break;
                  }
                  else if(byte_counter == 4)
                  {
                        sensor_buffer[1] = byte; // ultrasonic quadrant byte
                        byte_counter += 1;
                        break;
                  }
                  else if(byte_counter == 5)
                  {
                    sensor_buffer[2] = byte; // First distance byte
                    byte_counter += 1;
                    break;
                }
                else if(byte_counter == 6)
                {
                    sensor_buffer[3] = byte; // 2nd distance byte
                    sensor_buffer[4] = 0;    // placeholder in last array space
                    byte_counter += 1;
                    memset(receive_buffer,'0',sizeof(receive_buffer));
                }
                break;

            default:
                wait_for_sending_to_finish();
                delay_ms(200);

                break;
        }
}

void check_for_new_bytes_received()
{
        while(serial_get_received_bytes(UART1) != receive_buffer_position)
        {
            // Process the new byte that has just been received.
            process_received_byte(receive_buffer[receive_buffer_position]);

            // Increment receive_buffer_position, but wrap around when it gets to
            // the end of the buffer.
            if (receive_buffer_position == sizeof(receive_buffer)-1)
            {
                receive_buffer_position = 0;
            }
            else
            {
                receive_buffer_position++;
            }
        }
}

int main()
{


        servos_start(demuxPins, sizeof(demuxPins));

        // Set the servo speed to 150.  This means that the pulse width
        // will change by at most 15 microseconds every 20 ms.  So it will
        // take 1.33 seconds to go from a pulse width of 1000 us to 2000 us.
        set_servo_speed(0, init_speed);
        set_servo_speed(1, init_speed);
        set_servo_speed(2, init_speed);
        set_servo_speed(3, init_speed);

        // Make all the servos go to a neutral position.
        set_servo_target(0, shoulder_dn);     //shoulder up/down
        set_servo_target(1, 1500);      //elbow
        set_servo_target(2, neutral_servo_pos);     //shoulder rotation
        set_servo_target(3, hand_open);             //hand

        clear();     // clear the LCD
        print("Send serial");
        lcd_goto_xy(0, 1); // go to start of second LCD row
        print("or press Btn");

        // Set the baud rate to 9600 bits per second.  Each byte takes ten bit
        // times, so you can get at most 960 bytes per second at this speed.
        serial_set_baud_rate(UART1, 9600);

        // Start receiving bytes in the ring buffer.
        serial_receive_ring(UART1, receive_buffer, sizeof(receive_buffer));

    while(1)
    {
            // USB_COMM is always in SERIAL_CHECK mode, so we need to call this
            // function often to make sure serial receptions and transmissions
            // occur.
            serial_check();

            check_for_new_bytes_received();
            // Deal with any new bytes received unless we have a complete sample
            if (byte_counter >= 7)
            {

                    if (sensor_buffer[1] == '0')                //If zero, no sector detected
from sensors
                    {
                            clear();                            // clear the LCD
                            lcd_goto_xy(0, 1);                  // go to start of second LCD
row
                            print("Nada: ");
                            set_servo_speed(0, init_speed);
                            set_servo_speed(1, init_speed);
                            set_servo_speed(2, init_speed);
                            set_servo_speed(3, init_speed);

                            // Make all the servos go to a neutral position.
                            set_servo_target(0, shoulder_dn);           //shoulder up/down
                            set_servo_target(1, 1500);                  //elbow
                            set_servo_target(2, neutral_servo_pos);     //shoulder rotation
                            set_servo_target(3, hand_closed);           //hand
                            delay_ms(1000);
                            memset(receive_buffer,'0',sizeof(receive_buffer));
                            byte_counter = 0;  //reset counter if nothing is in front of the
robot
                    }
                    else if (sensor_buffer[1] == '1')                   //sector value, 1-6
                    {
                            clear();                        // clear the LCD
                            lcd_goto_xy(0, 1);              // go to start of second LCD row
                            print("Q1: ");
                            delay_ms(500);

                            set_servo_speed(0, init_speed);
                            set_servo_speed(1, init_speed);
```

```
                set_servo_speed(2, init_speed);
                set_servo_speed(3, init_speed);

                // Make all the servos go to a neutral position.
                set_servo_target(0, shoulder_up);     //shoulder up/down
                set_servo_target(1, 2000);            //elbow 1/2 up
                set_servo_target(2, 1375);            //shoulder rotation Left 1/4
                set_servo_target(3, hand_open);             //hand
                delay_ms(500);
                memset(receive_buffer,'0',sizeof(receive_buffer));
                byte_counter = 0;
        }
        else if (sensor_buffer[1] == '2')
        {
                clear();                    // clear the LCD
                lcd_goto_xy(0, 1);          // go to start of second LCD row
                print("Q2: ");
                delay_ms(500);

                set_servo_speed(0, init_speed);
                set_servo_speed(1, init_speed);
                set_servo_speed(2, init_speed);
                set_servo_speed(3, init_speed);

                // Make all the servos go to a neutral position.
                set_servo_target(0, shoulder_up);     //shoulder up/down
                set_servo_target(1, 2000);      //elbow 1/2 up
                set_servo_target(2, 1450);      //shoulder rotation 1/2 Left
                set_servo_target(3, hand_open);             //hand
                delay_ms(500);
                memset(receive_buffer,'0',sizeof(receive_buffer));
                byte_counter = 0;

        }
        else if(sensor_buffer[1] == '3')
        {
                clear();                    // clear the LCD
                lcd_goto_xy(0, 1);          // go to start of second LCD row
                print("Q3 ");
                delay_ms(500);

                set_servo_speed(0, init_speed);
                set_servo_speed(1, init_speed);
                set_servo_speed(2, init_speed);
                set_servo_speed(3, init_speed);

                // Make all the servos go to a neutral position.
                set_servo_target(0, shoulder_up);     //shoulder up/down
                set_servo_target(1, 1750);      //elbow 1/4 up
                set_servo_target(2, 1375);      //shoulder rotation 1/4 Left
                set_servo_target(3, hand_open);             //hand
                delay_ms(500);
                memset(receive_buffer,'0',sizeof(receive_buffer));
                byte_counter = 0;

        }
        else if (sensor_buffer[1] == '4')
        {
```

```
                        clear();                        // clear the LCD
                        lcd_goto_xy(0, 1);      // go to start of second LCD row
                        print("Q4: ");
                        delay_ms(500);

                        set_servo_speed(0, init_speed);
                        set_servo_speed(1, init_speed);
                        set_servo_speed(2, init_speed);
                        set_servo_speed(3, init_speed);

                        // Make all the servos go to a neutral position.
                        set_servo_target(0, shoulder_up);           //shoulder up/down
                        set_servo_target(1, 1750);                  //elbow 1/4 up
                        set_servo_target(2, neutral_servo_pos);     //shoulder rotation 1/2
Left

                        set_servo_target(3, hand_open);       //hand
                        delay_ms(500);
                        memset(receive_buffer,'0',sizeof(receive_buffer));
                        byte_counter = 0;
                }
                else
                {
                        byte_counter = 0;
                        memset(receive_buffer,'0',sizeof(receive_buffer));
                }

                if (sensor_buffer[0] == '1')          // object detected in front of hand
                {
                        delay_ms(1000);
                        set_servo_speed(3, init_speed);
                        set_servo_target(3, hand_closed);     // close hand for handshake
                                                        // objective complete!
                        byte_counter = 0;
                        delay_ms(4000);
                        sensor_buffer[1] = '0';

                }
        }       // If the user presses the middle button, send "Robots Rule!"
                // and wait until the user releases the button.
        if (button_is_pressed(MIDDLE_BUTTON))
        {
                wait_for_sending_to_finish();
                memcpy_P(send_buffer, PSTR("Robots Rule!\r\n"), 12);
                serial_send(UART1, send_buffer, 12);
                send_buffer[11] = 0;                    // terminate the string
                clear();                        // clear the LCD
                lcd_goto_xy(0, 1);              // go to start of second LCD row
                print("TX: ");
                print(sensor_buffer);

                delay_ms(2000);
                byte_counter = 0;               // reset detect cycle by pressing button

                // Wait for the user to release the button.
                wait_for_button_release(MIDDLE_BUTTON);
        }
    }
}
```

Servo Control

Servos are fairly simple to use, just give them 3.3V to 6V and a control signal and the arm moves to a position. Most have a range of movement of 180°, with the control signal pulse running at 50 Hz and 1-2 ms ‘high’ time. Changing the pulse signal changes the arm position. 1ms is one extreme, 2ms is the other, and 1.5 ms pulses put the arm about in the middle.



Servo in Middle Position

Sensor Development

Purpose

The sensors are used to locate the hand (target object) so that the arm can shake hands with someone. The sensor network was developed to cover as large of an area as possible with sufficient resolution.

Design

Several methods of sensing the target object were considered. The choices were narrowed to two different sensor types, and the first choice was IR beam sensors.

IR Sensors

IR sensors are active devices that send out a pulse of light in the Infrared (IR) wavelength. Several types were tested with various distance specifications. Ultimately though, range was not the issue. In fact they were fairly accurate. The issue turned out to be the beam width / coverage area. The beam was very narrow, and while ensuring accuracy, was not beneficial for our application. The sensor array needed to cover as much area as possible while still giving fairly accurate depth measurements. So we continued searching, which brought us to Acoustic Sensors.

Acoustic Sensors

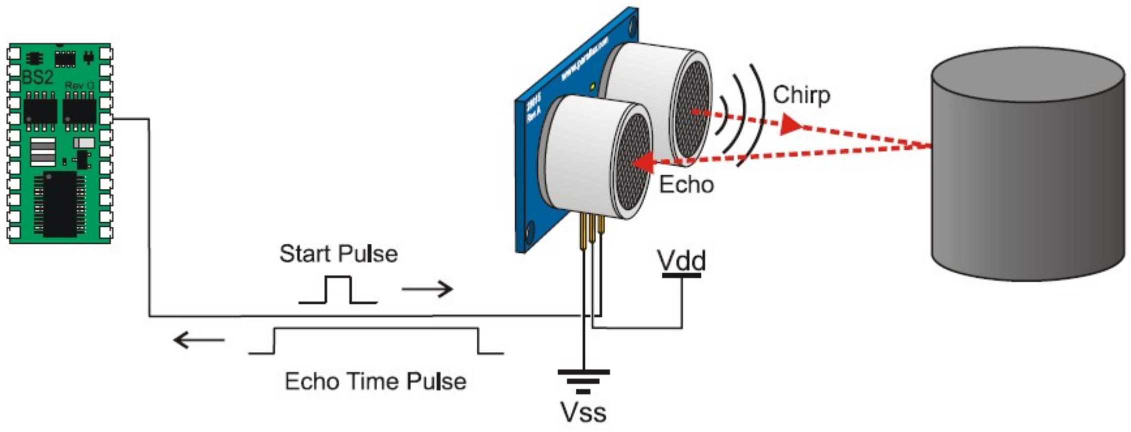


Figure 8

It was determined that acoustic sensors would have the best combination of accuracy in the depth axis and yet sufficient coverage in cross sectional area. Several layouts were considered, including square pattern, diamond and tight diamond.

The final pattern selected was a square pattern, which would, in theory, allow the area to be divided into 9 distinct zones or locations. The arm would be targeted at those areas.

First design

The sensors were laid out in a square pattern, covering an area of about 0.3 meters. The intent was that the area would be differentiated into 9 distinct zones. The zones were determined by a search algorithm based up on the sensor or combination of sensors that detected an object. In the algorithm it was first determined if an object was within the bounds of the region of interest. If an object was found, it was next determined a) which sensor measured the closest measurement and b) whether any other sensors detected the same object. This was determined by filtering other sensors based upon the difference in measurement compared to the sensor that had the nearest measurement. Any sensors that had measurements outside of this bound were rejected.

Once the map of activated sensors was completed the search algorithm was performed to determine position of the object. The position was determined based on the sensor or combination of sensors detecting the object. The original code is shown below.

The sensors were laid out in the pattern. The design worked well under most conditions.

During testing of the first design it was found that there was one drawback which had not been foreseen. When a human hand is straight on with respect to the sensor, with fingers fully extended, there is not enough surface area to provide a return signal to the sensor. Also, we suspect that acoustic background noise found in the room was causing interference with the sensors as well.

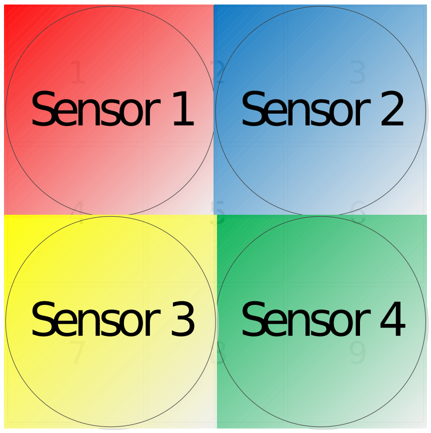


Figure 9

<u>Original Search Algorithm (for square pattern in Figure 1).</u>

If Sensor 1 only than object is in sector 1.

If Sensor 1 and Sensor 2 than object is in sector 2.

if Sensor 2 only than object is in sector 3.

If Sensor 1 and Sensor 3 than object is in sector 4.

If Sensor 1, Sensor 2, Sensor 3 and Sensor 4 than object is in sector 5.

If ANY THREE of the four sensors are triggered, then the object is ASSUMED to be sector 5.

If sensor 1 and 4 are triggered OR sensors 2 and 3 are triggered, than it is assumed the object is in position 5.

If Sensor 2 and Sensor 4 than object is in sector 6.

If Sensor 3 only than object is in sector 7.

If Sensor 3 and Sensor 4 than object is in sector 8.

if Sensor 4 only than object is in sector 9.

<u>Second Design</u>

To compensate for this problem it was decided that the sensors would be aimed inward at 60° angle off center. This would allow the sensors to catch object from the side and hopefully connecting with more surface area.

The results were mixed. There seemed to be some improvement, but the system persisted to have sensitivity issues. This reduced the number of zones that could be covered to 4 (figure 6). It was attempted to gain back 2 zones which would have brought the total to 6, but due to interaction of the sensor grid it was not feasible. The sensors were now near sighted at the center region due to the fact that the grid was aimed inward (Figure 3).

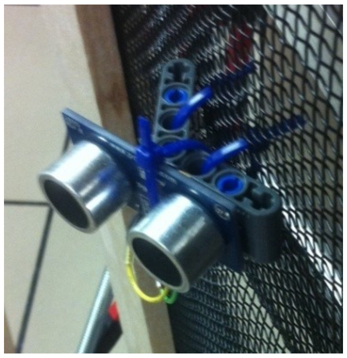
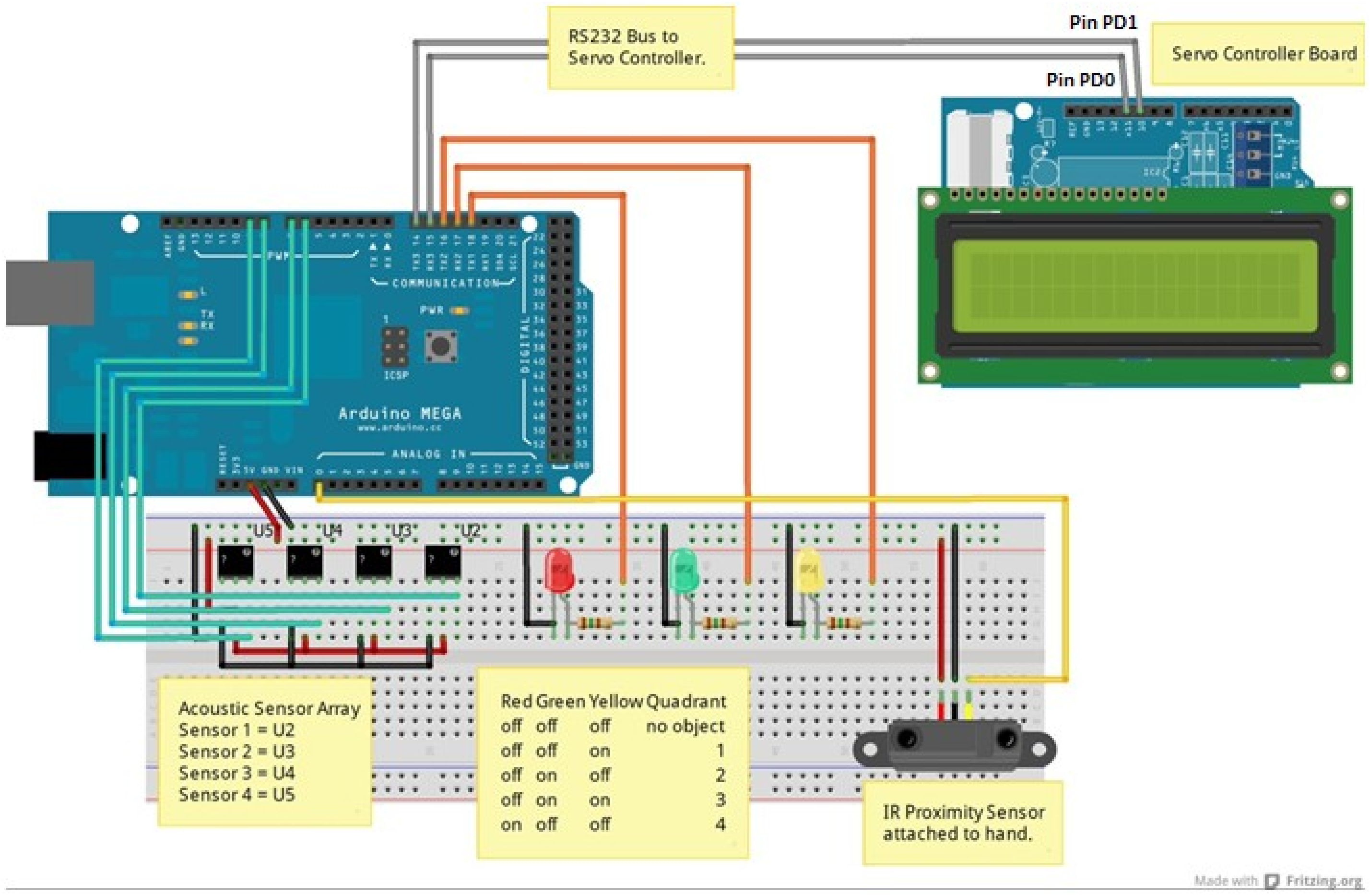
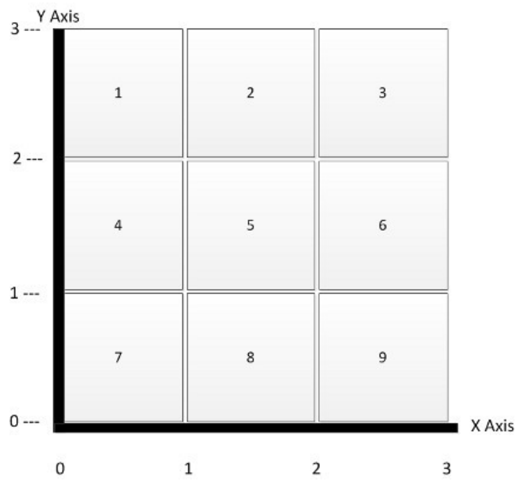
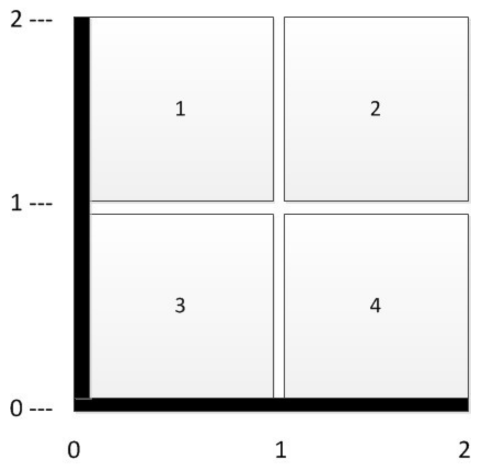


Figure 10



Red	Green	Yellow	Quadrant
off	off	off	no object
off	off	on	1
off	on	off	2
off	on	on	3
on	off	off	4





<u>Code for first version of sensors</u>

```
#include <SoftwareSerial.h>

/* Ping))) Sensor

   This sketch reads a PING))) ultrasonic rangefinder and returns the
   distance to the closest object in range. To do this, it sends a pulse
   to the sensor to initiate a reading, then listens for a pulse
   to return.  The length of the returning pulse is proportional to
   the distance of the object from the sensor.

   The circuit:
    * +V connection of the PING))) attached to +5V
    * GND connection of the PING))) attached to ground
    * SIG connection of the PING))) attached to digital pin 7

   http://www.arduino.cc/en/Tutorial/Ping

   created 3 Nov 2008
   by David A. Mellis
   modified 30 Aug 2011
   by Tom Igoe

   This example code is in the public domain.

 */

// this constant won't change.  It's the pin number
// of the sensor's output:
#define rxPin 15
#define txPin 14
SoftwareSerial myserial = SoftwareSerial(rxPin,txPin);


const int SensPins[] = {6,7,8,9};
const int opticSense = 0;
```

```
//const unsigned int nAverages = 10;
const unsigned int senseCount = 4;
const unsigned long innerBoundary = 25;
const unsigned long outerBoundary = 65;
int verbose = 0;
//static unsigned int duration_all_avgbin[senseCount][nAverages];

//static unsigned int count = 0;


int dataMode = 2;           // dataMode = 0: Raw data output in string serial out.
                            // dataMode = 1: X/Y grid in string serial out.
                            // dataMode = 2: X/Y grid in byte form on SPI (Untested).



// boolean check if object in target area.
                            // Will trip sensors if ANY object within 110cm.
                            // Closest point of interest.
                            // mid point +/- value.  3 zones.
//long checkRangeTemp = 0;
void setup() {
  // initialize serial communication:
  Serial.begin(9600);
  //myserial(10,11);
  myserial.begin(115200);
  analogReference(DEFAULT);
}
void loop()
{
  // establish variables for duration of the ping,
  // and the distance result in inches and centimeters:
  //long duration1, duration2, duration3, inches1, inches2, inches3;
 // boolean thisSensorActive[senseCount] = {false};

 /*if (myserial.available())
 {
      unsigned char test = (unsigned char)mySerial.read();
      //Serial.(mySerial.read());
 }
  if (Serial.available())
    mySerial.write(Serial.read());
    */

  boolean objectInRange = false;
  boolean foundTarget = false;
  unsigned int senseBin_cm[senseCount] = {0};
  unsigned long duration_all_tier[senseCount] ={0};
  unsigned long closestObject = 0;
  //byte closestObject = 00;
  unsigned long nearestBound = 7;
  unsigned long tempCompare = 0;
  unsigned int opticSensor = 0;
  unsigned int optiDetect = 0;
  //unsigned long tempHolder = 0;
  int nearestPin = 0;
  int xGrid = 0, yGrid = 0;
  byte activeSensor = B0000;
  // The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
for(int i = 0; i < senseCount; i++){
    pinMode(SensPins[i], OUTPUT);
    digitalWrite(SensPins[i], LOW);
    delayMicroseconds(2);
    digitalWrite(SensPins[i], HIGH);
    delayMicroseconds(5);
    digitalWrite(SensPins[i], LOW);
    pinMode(SensPins[i], INPUT);
    duration_all_tier[i] = pulseIn(SensPins[i], HIGH);
    delayMicroseconds(3000);

  }

  for(int i = 0; i < senseCount; i++){
    //senseBin_cmAvg[i] = microsecondsToCentimeters(aveMeaasuredValue[i]);
    unsigned long tempHolder = microsecondsToCentimeters(duration_all_tier[i]);
    senseBin_cm[i] = (unsigned int)tempHolder;
    //senseBin_cm[i] = (unsigned int)microsecondsToCentimeters(duration_all_tier[i]);
  }
opticSensor = analogRead(opticSense);
//*******************************************************************
  /*

  */
  // The same pin is used to read the signal from the PING))): a HIGH
  // pulse whose duration is the time (in microseconds) from the sending
  // of the ping to the reception of its echo off of an object.



  // convert the time into a distance

  // Search for hand / object within zone.
  // Checks for object within zone boundary.

for(int i = 0; i < senseCount; i++){
    tempCompare = constrain(senseBin_cm[i],innerBoundary,outerBoundary);
    if(tempCompare == senseBin_cm[i])
    {
      objectInRange = true;
    }
  }
  tempCompare = 0; // reset temp compare to 0;
  // If there is an object in range, find nearest object and sensor detected on.
  if(objectInRange)
  {
    closestObject = senseBin_cm[0];
    for(int i = 0; i < (senseCount - 1); i++){
      closestObject = min(closestObject,senseBin_cm[i+1]);
    }
    for(int i = 0; i < senseCount; i++)
    {
      if(closestObject == senseBin_cm[i])
      {
         nearestPin = SensPins[i];
      }
    }
    for(int i = 0; i < senseCount; i++){
      if(nearestPin == SensPins[i])
      {
        bitSet(activeSensor,i);
      }
      if(nearestPin != SensPins[i])
      {
        tempCompare = constrain(senseBin_cm[i],closestObject,
        (closestObject + nearestBound));
    if(tempCompare == senseBin_cm[i])
    {
      bitSet(activeSensor,i);
    }
      }
    }
    foundTarget = true;
}

  // Determine the x / y grid based on sensors that see object.
  // Sensor layout from perspective of facing robot.
  //
  //      Sensor 1 (pin 7)        Sensor 2 (pin 8)
  //
  //      Sensor 3 (pin 9)        Sensor 4 (pin 10)
  // **********************************************************
  if(foundTarget)
  {
    if(activeSensor == B1111 ||activeSensor == B1110 ||
    activeSensor == B1101 || activeSensor == B1011 ||
    activeSensor == B0111)
    {
      xGrid = 2;
      yGrid = 2;
    }
    //  thisSensorActive[0] && thisSensorActive[1]
    else if(activeSensor == B0011)
    {
       xGrid = 2;
       yGrid = 3;
    }
    //thisSensorActive[0] && thisSensorActive[2]
    else if(activeSensor == B0101)
    {
       xGrid = 1;
       yGrid = 2;
    }
    //thisSensorActive[1] && thisSensorActive[3]
    else if(activeSensor == B1010)
    {
       xGrid = 3;
       yGrid = 2;
    }
    //thisSensorActive[2] && thisSensorActive[3]
    else if(activeSensor == B1100)
    {
       xGrid = 2;
       yGrid = 1;
    }
    //thisSensorActive[0]
    else if(activeSensor == B0001)
    {
       xGrid = 1;
       yGrid = 3;
    }
    //thisSensorActive[1]
    else if(activeSensor == B0010)
    {
       xGrid = 3;
       yGrid = 3;
    }
    //thisSensorActive[2]
    else if(activeSensor == B0100)
    {
       xGrid = 1;
       yGrid = 1;
    }
    //thisSensorActive[3]
    else if(activeSensor == B1000)
    {
       xGrid = 3;
       yGrid = 1;
    }
    if(opticSensor > 600)
    {
      optiDetect = 1;
    }
    else
    {
      optiDetect = 0;
    }
  }

 //if(verbose)  // Verbose is a raw data troubleshooting mode.
  switch (dataMode) {
    case 0:
      Serial.print("Sensor 1: ");
      Serial.print(senseBin_cm[0]);
      Serial.print("cm: Optic: ");
      Serial.print(opticSensor);
      Serial.print("\n");
      Serial.print("Sensor 2: ");
      Serial.print(senseBin_cm[1]);
      Serial.print("cm: Optic: ");
      Serial.print(opticSensor);
      Serial.print("\n");
      Serial.print("Sensor 3: ");
      Serial.print(senseBin_cm[2]);
      Serial.print("cm: Optic: ");
      Serial.print(opticSensor);
      Serial.print("\n");
      Serial.print("Sensor 4: ");
      Serial.print(senseBin_cm[3]);
      Serial.print("cm: Optic: ");
      Serial.print(opticSensor);
      Serial.print("\nNearest object: ");
      Serial.print(closestObject);
      Serial.println();
      break;
case 1:
      Serial.print("Distance: ");
      Serial.print(closestObject);
      Serial.print("\n");
      Serial.print("X Grid: ");
      Serial.print(xGrid);
      Serial.print("\n");
      Serial.print("Y Grid: ");
      Serial.print(yGrid);
      break;
    case 2:
    unsigned char out1 = (unsigned char)xGrid;
    unsigned char out2 = (unsigned char)yGrid;
    //if(
    unsigned char out3 = (unsigned char)closestObject;
    unsigned char out4 = (unsigned char)optiDetect;
      Serial.print(":::");
      myserial.print(":::");
      Serial.print(optiDetect);
      myserial.print(out3);
       Serial.print(":");
      Serial.print(xGrid);
      myserial.print(out1);

      //myserial.print("A");
      Serial.print(":");
      //myserial.print(":");
      Serial.print(yGrid);
      myserial.print(out2);
      Serial.print(":");
      //myserial.print(":");
if(closestObject < 10)
      {
        Serial.print("0");
        Serial.print(closestObject);
        myserial.print("0");
        myserial.print(out2);
      }
      else if(closestObject >= 10)
      {
        Serial.print(closestObject);
        myserial.print(out2);
      }
      //Serial.print(":");
      //myserial.print(":");
```

```
        //Serial.print(myserial.read());
        Serial.print("\n");
        //myserial.print("\n");
         Serial.println();
        //myserial.println();
        break;
    }
    //delay(100);
    delayMicroseconds(2000000);
}
long microsecondsToCentimeters(long microseconds)
{
  // The speed of sound is 340 m/s or 29 microseconds per centimeter.
  // The ping travels out and back, so to find the distance of the
  // object we take half of the distance travelled.
  return microseconds / 29 / 2;
}
```

Final version of code for sensors

```
#include <SoftwareSerial.h>

/* Ping))) Sensor

   This sketch reads a PING))) ultrasonic rangefinder and returns the
   distance to the closest object in range. To do this, it sends a pulse
   to the sensor to initiate a reading, then listens for a pulse
   to return.  The length of the returning pulse is proportional to
   the distance of the object from the sensor.

   The circuit:
    * +V connection of the PING))) attached to +5V
    * GND connection of the PING))) attached to ground
    * SIG connection of the PING))) attached to digital pin 7

   http://www.arduino.cc/en/Tutorial/Ping

   created 3 Nov 2008
   by David A. Mellis
   modified 30 Aug 2011
   by Tom Igoe

   This example code is in the public domain.

 */

// this constant won't change.  It's the pin number
// of the sensor's output:
#define rxPin 2
#define txPin 3
```

```
#define ledYellow 13
#define ledRed 12
#define ledGreen 11
SoftwareSerial myserial = SoftwareSerial(rxPin,txPin);


const int SensPins[] = {6,7,8,9};
const int opticSense = 0;
const unsigned int senseCount = 4;
const unsigned long innerBoundary = 46;
const unsigned long outerBoundary = 55;

 int nearestPin = 0;

int dataMode = 2;          // dataMode = 0: Raw data output in string serial out.
                           // dataMode = 1: X/Y grid in string serial out.
                           // dataMode = 2: X/Y grid in byte form on SPI (Untested).



// boolean check if object in target area.
                           // Will trip sensors if ANY object within 110cm.
                           // Closest point of interest.
                           // mid point +/- value.  3 zones.
//long checkRangeTemp = 0;
void setup() {
  // initialize serial communication:
  Serial.begin(9600);
  //myserial(10,11);
  myserial.begin(9600);
  analogReference(DEFAULT);
  pinMode(ledYellow, OUTPUT);
  pinMode(ledRed, OUTPUT);
  pinMode(ledGreen, OUTPUT);
}
void loop()
{
  // establish variables for duration of the ping,
  // and the distance result in inches and centimeters:

  boolean objectInRange = false;
  boolean foundTarget = false;
  unsigned int senseBin_cm[senseCount] = {0};
  unsigned long duration_all_tier[senseCount] ={0};
  unsigned long closestObject = 0;
  unsigned long nearestBound = 7;
  unsigned long tempCompare = 0;
  unsigned int opticSensor = 0;
  unsigned int optiDetect = 0;
  int coordLocation = 0;
  byte activeSensor = B0000;
  // The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:

  for(int i = 0; i < senseCount; i++){
    pinMode(SensPins[i], OUTPUT);
    digitalWrite(SensPins[i], LOW);
    delayMicroseconds(2);
    digitalWrite(SensPins[i], HIGH);
    delayMicroseconds(5);
    digitalWrite(SensPins[i], LOW);
    pinMode(SensPins[i], INPUT);
    duration_all_tier[i] = pulseIn(SensPins[i], HIGH);
    delayMicroseconds(3000);

  }

  for(int i = 0; i < senseCount; i++){
    unsigned long tempHolder = microsecondsToCentimeters(duration_all_tier[i]);
    senseBin_cm[i] = (unsigned int)tempHolder;
  }
opticSensor = analogRead(opticSense);
//*****************************************************************
  /*

  */
  // The same pin is used to read the signal from the PING))): a HIGH
  // pulse whose duration is the time (in microseconds) from the sending
  // of the ping to the reception of its echo off of an object.



  // convert the time into a distance
for(int i = 0; i < senseCount; i++){
    tempCompare = constrain(senseBin_cm[i],(unsigned int)innerBoundary,(unsigned
int)outerBoundary);

    if(tempCompare == senseBin_cm[i])
    {
      objectInRange = true;
    }
  }
    if(objectInRange)
    {
    closestObject = senseBin_cm[0];
    for(int i = 0; i < (senseCount - 1); i++){
      closestObject = min(closestObject,senseBin_cm[i+1]);
    }

    for(int i = 0; i < senseCount; i++)
    {
      if(closestObject == senseBin_cm[i])
      {
         nearestPin = SensPins[i];
      }
    }
    for(int i = 0; i < senseCount; i++){
      if(nearestPin == SensPins[i])
      {
        bitSet(activeSensor,i);
        foundTarget = true;
      }
    }
    }

  tempCompare = 0; // reset temp compare to 0;
// If there is an object in range, find nearest object and sensor detected on.

  // Determine the x / y grid based on sensors that see object.
  // Sensor layout from perspective of facing robot.
  //
  //      Sensor 1 (pin 7)        Sensor 2 (pin 8)
  //
  //      Sensor 3 (pin 9)        Sensor 4 (pin 10)
  // **********************************************************
  //
  //      Quadrant 1      Quadrant 2
  //
  //      Quadrant 3      Quadrant 4
  // **********************************************************
  if(foundTarget)
  {
    if(activeSensor == B0001)
    {
       //xGrid = 2;
       //yGrid = 2;
       coordLocation = 2;
       digitalWrite(ledYellow, HIGH);
       digitalWrite(ledRed, LOW);
       digitalWrite(ledGreen, HIGH);
       delay(1000);
       digitalWrite(ledYellow, LOW);
       digitalWrite(ledRed, LOW);
       digitalWrite(ledGreen, LOW);
    }
    //thisSensorActive[1]
    else if(activeSensor == B0010)
    {
       //xGrid = 1;
       //yGrid = 2;
       coordLocation = 1;
       digitalWrite(ledYellow, HIGH);
       digitalWrite(ledRed, LOW);
       digitalWrite(ledGreen, LOW);
       delay(1000);
       digitalWrite(ledYellow, LOW);
       digitalWrite(ledRed, LOW);
       digitalWrite(ledGreen, LOW);
    }
    //thisSensorActive[2]
    else if(activeSensor == B0100)
    {
       //xGrid = 2;
       //yGrid = 1;
       coordLocation = 4;
       digitalWrite(ledYellow, LOW);
       digitalWrite(ledRed, HIGH);
       digitalWrite(ledGreen, HIGH);
```

```
      delay(1000);
      digitalWrite(ledYellow, LOW);
      digitalWrite(ledRed, LOW);
      digitalWrite(ledGreen, LOW);
    }
    //thisSensorActive[3]
    else if(activeSensor == B1000)
    {
      //xGrid = 1;
      //yGrid = 1;
      coordLocation = 3;
      digitalWrite(ledYellow, LOW);
      digitalWrite(ledRed, HIGH);
      digitalWrite(ledGreen, LOW);
     delay(1000);
      digitalWrite(ledYellow, LOW);
      digitalWrite(ledRed, LOW);
      digitalWrite(ledGreen, LOW);
    }
    if(opticSensor > 600)
    {
     optiDetect = 1;
    }
    else
    {
     optiDetect = 0;
    }
  }

  if(closestObject >= 100)
  {
    closestObject = 99;
  }

 //if(verbose)  // Verbose is a raw data troubleshooting mode.
  switch (dataMode) {
    case 2:
    unsigned char out1 = (unsigned char)coordLocation;
    //unsigned char out2 = (unsigned char)yGrid;
    unsigned char out3 = (unsigned char)closestObject;
    unsigned char out4 = (unsigned char)optiDetect;
      Serial.print(":::");
      myserial.print("xxx");
      //delay(50);
      Serial.print(optiDetect);
      myserial.print(out4);
      //delay(50);
       Serial.print(":");
      Serial.print(coordLocation);
      myserial.print(out1);
      //delay(50);
      Serial.print(":");
      if(closestObject < 10)
      {
        Serial.print("0");
        Serial.print(closestObject);
        myserial.print("0");
        //delay(50);
        myserial.print(out3);
        //delay(50);
      }
      else if(closestObject >= 10)
      {
        Serial.print(closestObject);
        myserial.print(out3);
        //delay(50);
      }
      Serial.print("\n");
      Serial.println();
      break;
  }
  //delay(100);
  delay(2000);
}

long microsecondsToCentimeters(long microseconds)
{
  // The speed of sound is 340 m/s or 29 microseconds per centimeter.
  // The ping travels out and back, so to find the distance of the
  // object we take half of the distance travelled.
  return microseconds / 29 / 2;
}
```

Proposed Improvements / Replacements and Conclusion

If a running average of the measurements were maintained this would provide statistic information to determine where the object is and whether or not it wanted to shake hands with us.

Parts List

Most of the arm construction custom made using general materials. The main purchases were the control boards and the sensors.

Item	Quantity	Cost ($)	Subtotal
Orangutan SVP 1284 - Robot Controller	1	100	100
Arduino Mega - Sensor Control Board	1	60	60
Arm Lamp	1	5	5
Assorted Nuts and Bolts	1	6	6
Drills, Pliers and Assorted Tools	Many	~	~
AA Battery Pack	2	3	6

Hobby Mending Plate Pack	1	4	4
High-Torque Servos	4	10	40
Ping))) Ultrasonic Sensor	4	30	120
Optical Sensor	1	12	12
		Total	**353**

Project Conclusions

The task of designing and implementing the Handshaking Robot Arm was an instructive experience. Through study of servo motor control, state machines, sensor interfacing, and hard work the arm is able to detect a user approaching and move to shake hands. The Pololu Orangutan was an excellent fit for the features our project required. The onboard servo controllers easily allowed control of the arm motion. One of our boards had a power problem, so an Arduino Mega board was used to control the sensors. We chose to decouple the arm control and sensor portions of the project and communicate through a serial interface.

We learned how to modify a lamp arm in a way that allowed the servo motors freedom of turning while securing them in a protective and low-torque frame. We also learned that not all servo motors respond the same way… some do not have the internal stop devices. They spin one direction until the control pulse passes a halfway point, at which time they spin the opposite direction.

The final result of the project met our initial goals. We had a handshaking robot arm that detected an approaching hand and moved the arm via servos to the desired position. Some of the problems we encountered along the way were: noise that interfered with our sensors; plastic servo arms breaking; torque issues and servo limitations, overcome by counterbalanced springs.

References

Pololu site with reference documents and example programs for Orangutan SVP development board: http://www.pololu.com/catalog/product/1327

"Embedded Robotics," Braunl, Thomas ; 3rd Edition, 2008, Springer

"Artificial Intelligence," Luger, George F. ; 6th Edition, 2009, Pearson Education

Prof. Marek Perkowski, 'Embedded Robotics' class at Portland State,
http://web.cecs.pdx.edu/~mperkows/CLASS_479/index2.html

The documents are located at Michael's class webpage here:

https://sites.google.com/a/pdx.edu/classes-engstrom/mysite/project-home