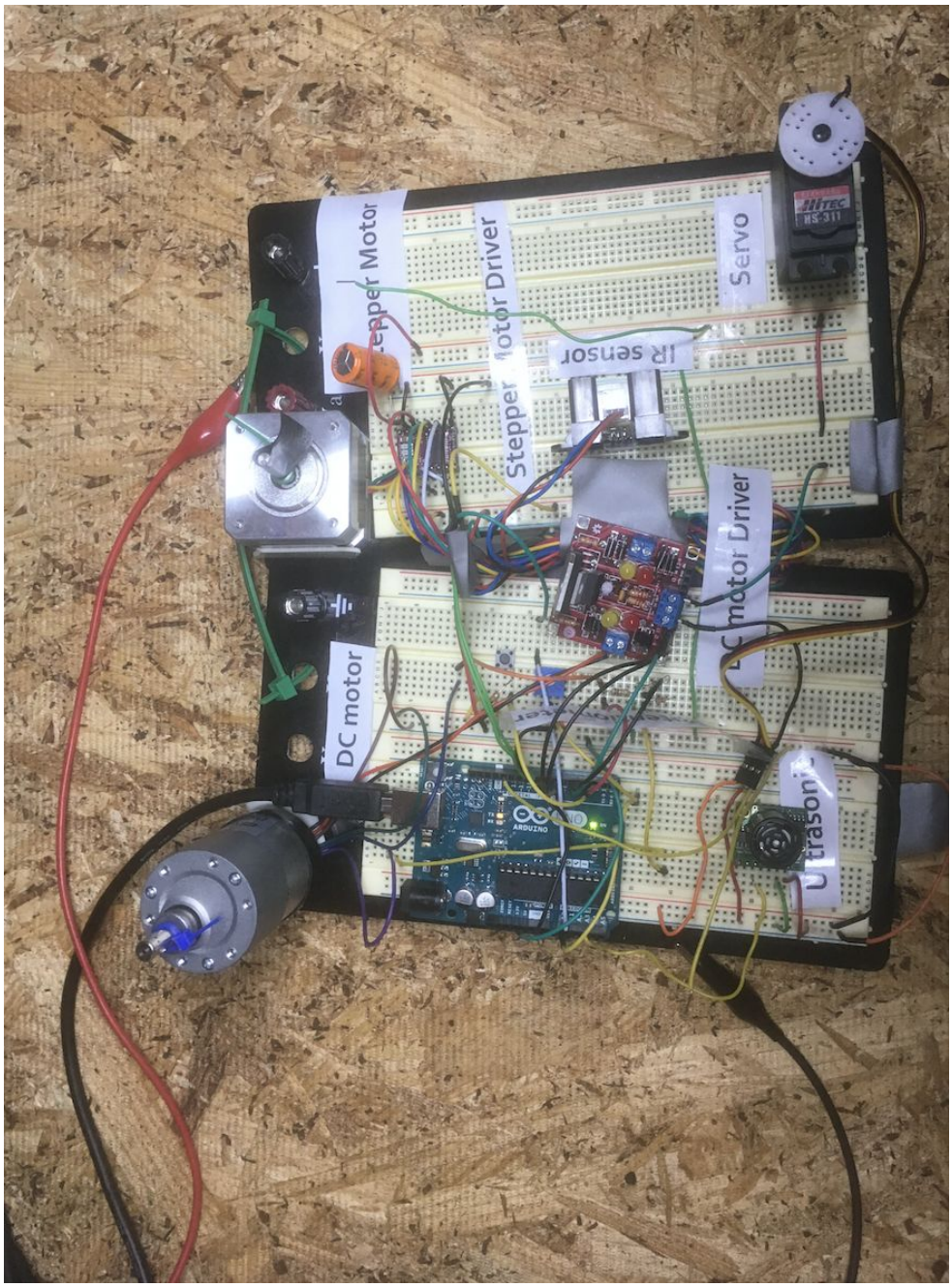
ILR 1 Sensors and Motors

Team Name: F
Team Members: Karthik, Sumit, Xiaoyang, Juncheng
Individual: Karthik



Overview

Goal of the project was to demonstrate the capability to operate and control 3 different motors through 3 different sensors and be able to override the controls using a UI. The following high level components/modules were used for this project:

1. DC motor & Potentiometer
2. Servo motor & Ultrasonic sensor
3. Stepper motor & infrared sensor
4. Arduino Uno microcontroller
5. Processing based UI

Individual Deliverables

Introduction

Following were the two individual deliverables:

1. Servo motor control through a ultrasonic sensor
2. Implement the infrastructure code to bring all sensors and motors together and interface with the UI

Servo motor control

For this project, a Hitech servo was used. Pins of a servo are connected as shown in Figure 1. Two of the pins are connected to the ground and 5V supply of arduino. The third pin is connected to one of the digital pins of arduino that can provide the pwn signal to the servo.

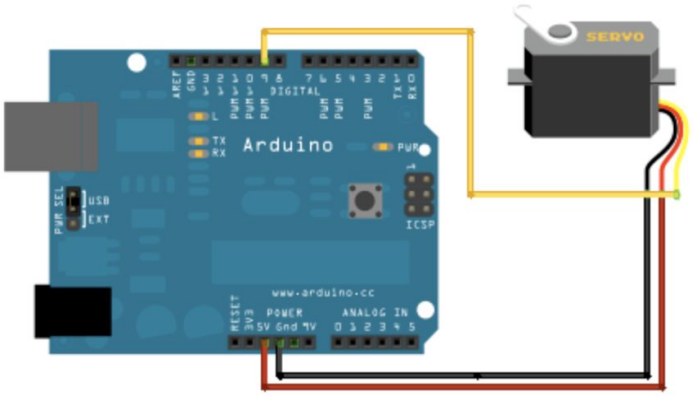


Figure 1.Hitech servo connection with Arduino Uno [1]

Ultrasonic sensor

The LV Max sonar ultrasonic sensor was used to control the servo motor. This sensor has a scale factor of Vcc/512. Since a voltage of 5V was used, the volts per inch was around 9mV/inch. Also, since the voltage

for the sonar was being read through the Analog to digital pin of the Arduino, which has a range of 1024, the final distance in inches was computed as Voltage/2

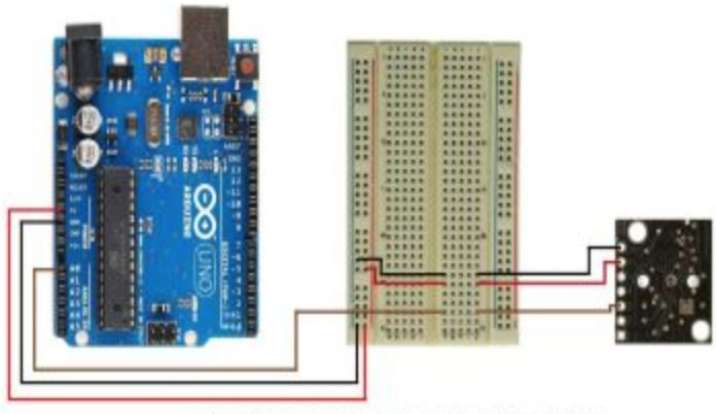


Figure 2. LV Max Sonar connection with Arduino Uno [2]

Software

Github : http://github.com/karamach/rescue_rangers

Individual deliverable for software was to write modules to control the servo using distances obtained from ultrasonic sensor and to write glue code to bring all components together. Key pieces of the software is explained below. The complete code is pasted in the Appendix.

Ultrasonic based Servo Motor control

The logic for computing the distance based on the ultrasound. As can be seen, the reading is obtained as an average of 5 samples and then halved to get the actual distance, to account for the fact that the analog pin generates an output between 0 to 1023 while the scale factor of the Ultrasonic sensor is Vcc/512.

```
// Get distance from Ultrasound
void updateDistance() {
  int sum = 0;
  int numSamples = 5;
  for (int i = 0; i < numSamples ; i++) {
    int volt = analogRead(ultraSoundState.pin);
    sum += volt;
    delay(5);
  }
  ultraSoundState.dist = (sum/numSamples) * 0.5;
}
```

The logic for computing the angle for the servo based on the distance is shown below. The servo is set at 0 degrees for distances less than 12 inches. For distances between 12 and 48 inches, it is linearly scaled to an angle between 0 and 180 degrees. For distances beyond 48 inches, it is set at 180 degrees.

```
// Compute servo angle.
// Servo is set to 0 for less than 12 inches
// Servo is set to a position determined as a linear function of distance between 12 and 48 inches
// Servo is set to 180 beyond 48 inches.
int computeAngle(int dist) {
  if (dist < 12) {
    return 0;
  } else if (dist > 48) {
    return 180;
  } else {
    return 5*(dist-12);
  }
}
```

Glue code

The glue code interfaces with the UI and the drives the components based on input from the UI. The format used for interaction between the UI and the glue code is shown below:

UI Command :
<begin_marker>,<motor_type>,<on/off>,<position>,<direction>,<velocity>,<end_marker>

Status message:
<begin_marker>,<on/off>,<position>,<direction>,<velocity>,<on/off>, …. <end_marker>
Note: the status is sent for each motor and sensor and the order is agreed upon.

The glue code is attached as an appendix.

Challenges

One of the key challenges in this deliverable was in implementing the glue code in a way that it could integrate control codes of different components with the UI. The control code for each of the components were written and designed independently and so had different semantics in terms of how they controlled the state. In hindsight, a better approach would have been for one team member to write most of the code except for very specific functions required for each motor. That would have brought in a lot more consistency in the overall code and would have resulted in far less integration time.

Progress

The MRSD project involves building an Autonomous Aerial System for assisting Search and Rescue Operations. The immediate deliverables for the milestone after finishing the CODR is pasted below:

10/20/2016
 	
Progress Review 1
	• Project Continuity
	  ○ Procure parts
	• Autonomous Flight System
	  ○ Ramp-up on Hexcopter operability
	• Sensing
	  ○ Finalize type and model of sensors to be used by conducting trade studies.
	  ○ Test/Validate sensors independently to ensure they meet specifications.

Status of tasks is listed below:

- Project Continuity
  - Procure parts
    - In discussion with Sponsor to get the Matrice 600 quadcopter and figure out logistics of operating it (at Sponsor place or at CMU)
- Autonomous Flight System
  - Ramp up on Hexcoter operability
    - Individual deliverable in progress:
      1. Have started looking at the various options of interacting with the DJI simulator (available only windows) and ROS based DJI API (available for Linux). Going through documentation and getting familiarized with the SDK.
- Sensing
  - Finalize type and model of sensors - Team has narrowed down 3 types of sensors to be used namely: thermal, RGB and sound.
    - Individual deliverables in progress:
      1. Evaluate exact range and sensitivity needed for the sound sensor so that the model and type can be decided on.
      2. Do some basic evaluation of how easy or difficult it is to do Voice Activity Detection from sound samples similar to ones that will be encountered in Aerial search and rescue scenarios.

References

[1] http://www.robotshop.com/blog/en/arduino-5-minute-tutorials-lesson-5-servo-motors-3636
[2] http://www.maxbotix.com/Ultrasonic_Sensors/MB1010.htm

Appendix

1-Servo Motor & Ultrasonic sensor code

svo_motor.h

```
#ifndef _SERVO_MOTOR_DRIVER_H_
#define _SERVO_MOTOR_DRIVER_H_

#include <Servo.h>

struct UltraSoundState {
  int MAX_DIST;
  int MIN_DIST;
  int pin;
  int dist;
  int on;
};

struct ServoState {
  int svo_pos;
  int on;
};

void setupServo();
void driveServo(int angle);
void driveServo();
void updateUSSensorState(int on);
void updateServoState(int on);

extern UltraSoundState ultraSoundState;
extern ServoState servoState;

#endif
```

svo_motor.cpp

```
#include "Arduino.h"
#include "svo_motor_driver.h"

Servo svo;
ServoState servoState;
UltraSoundState ultraSoundState;

// Setup
void setupServoUltrasound() {
  // Setup servo
  svo.attach(13);
  servoState.svo_pos = 0;
  servoState.on = 1;

  // Setup ultrasound
  ultraSoundState.MAX_DIST = 256;
  ultraSoundState.MIN_DIST = 6;
  ultraSoundState.pin = A3;
  ultraSoundState.dist = 0;
```

```
  ultraSoundState.on = 1;

}

// Servo state update
void updateServoState(int on) {
  servoState.on = on;
}

// Ultrasound state update
void updateUSSensorState(int on) {
  ultraSoundState.on = on;
}

// Get distance from Ultrasound
void updateDistance() {
  int sum = 0;
  int numSamples = 5;
  for (int i = 0; i < numSamples ; i++) {
    int volt = analogRead(ultraSoundState.pin);
    sum += volt;
    delay(5);
  }
  ultraSoundState.dist = (sum/numSamples) * 0.5;
}

// Drive servo to specified angle
void driveServo(int angle){
  if (servoState.on) {
    int currAngle = svo.read();
    if (currAngle > angle) {
      for (int svo_pos = currAngle; svo_pos >= angle; svo_pos -= 1) {
        svo.write(svo_pos);
      }
    } else {
      for (int svo_pos = currAngle; svo_pos <= angle; svo_pos += 1) {
        svo.write(svo_pos);
      }
    }
    delay(20);
    servoState.svo_pos = angle;
  }
}

// Compute servo angle.
// Servo is set to 0 for less than 12 inches
// Servo is set to a position determined as a linear function of distance between 12 and 48 inches
// Servo is set to 180 beyond 48 inches.
int computeAngle(int dist) {
  if (dist < 12) {
```

```
    return 0;
  } else if (dist > 48) {
    return 180;
  } else {
    return 5*(dist-12);
  }
}

// Drive servo based on distance.
void driveServo() {
  updateDistance();
  if ((servoState.on == 1) && (ultraSoundState.on == 1)) {
    driveServo(computeAngle(ultraSoundState.dist));
  }
}
```

2- Glue interface code

TeamF_Task7.ino

```
#include "svo_motor_driver.h"
#include "stepper_motor_driver.h"
#include "dc_motor_driver.h"

bool sensorMode;

void setup() {

  // Global mode
  sensorMode = false;

  // Setup motors
  DC_Initial();
  setupStepper();
  setupServo();

  // Initialize serial
  Serial.begin(9600);
  Serial.println("Ready");
}

// Servo, Stepper, DCMotor, US, IR, F
// State, Reading, Direction (0 F, 1 R), RPM (only for DC),
void writeStatus() {
  String statusMsg = "bom,";

  // Servo status
  statusMsg = statusMsg + servoState.on + "," + servoState.svo_pos + ",na,na,";

  // Stepper status
```

```
  statusMsg = statusMsg + stepperStatus.on_off + "," + stepperStatus.angle + ",na,na,";

  // DC status
  DC_Motor_Status dc_status;
  get_Motor_Status(dc_status);
  statusMsg = statusMsg + dc_status.onoff + "," + dc_status.degree + "," + dc_status.dir + "," + dc_status.vel + ",";

  // UltrasoundSensor status
  statusMsg = statusMsg + ultraSoundState.on + "," + ultraSoundState.dist + ",na,na,";

  // InfraredSensor status
  statusMsg = statusMsg + irSensorStatus.on_off + "," + irSensorStatus.Distance + ",na,na,";

  // Potentio,eter status
  statusMsg = statusMsg + "1," + dc_status.sensorValue + ",na,na," ;

  // Print msg
  if (sensorMode == true)
    statusMsg += "1";
  else
    statusMsg += "0";
  statusMsg += ",eom";
  Serial.println(statusMsg);
}

void processCommand(String commands[20], int numCommands) {
  if (numCommands == 0)
    return;

  if (commands[0].equals("sensor-based")) {
    sensorMode = true;
    return;
  } else if (commands[0].equals("gui-based")) {
    sensorMode = false;
    updateState(0, 0, 0, 0);
    return;
  } else {
    if (!sensorMode) {
      if (commands[0].equals("m_dcm")) {
        // bom,m_dcm,1,90,na,na,eom
        String on = commands[1];
        String deg = commands[2];
        String dir = commands[3];
        String vel = commands[4];
        if (on.toInt() == 1) {
          if (vel.equals("na"))
            updateState(3, 0, dir.toInt(), deg.toInt());
          else
            updateState(2, vel.toInt(), dir.toInt(), 0);
        } else {
```

```
          updateState(0, 0, 0, 0);
        }
        driveDCMotor();
      } else if (commands[0].equals("m_stp")) {
        // bom,m_stp,1,90,1,na,eom
        // bom,m_stp,1,90,0,na,eom
        int angle = commands[2].toInt();
        int dir = commands[3].toInt() == 1 ? 1: -1;
        updateStepperState(angle*dir, 200, 1);
        driveStepper();

      } else if (commands[0].equals("m_ser")) {
        // bom,m_ser,1,90,na,na,eom
        int on = commands[1].toInt();
        updateServoState(on);
        if (on == 1) {
          int angle = commands[2].toInt();
          driveServo(angle);
        }
      } else if (commands[0].equals("s_isr")) {
        int on = commands[1].toInt();
        irSensorStatus.on_off = on;
      } else if (commands[0].equals("s_usr")) {
        int on = commands[1].toInt();
        updateUSSensorState(on);
      }
    }
  }
}

int readCommand(String commands[20]) {
  int count = 0;
  if (Serial.available() > 0) {
    String input = Serial.readString();
    Serial.println("log.." + input);
    if (!input.startsWith("bom") || !input.endsWith("eom\n"))
      return 0;
    input.trim();
    //Serial.println("log.." + input);

    String currString = "";
    int currPos = input.indexOf(',', 0) + 1;
    while (!currString.equals("eom")) {
      int pos = input.indexOf(',', currPos);
      if (pos != -1) {
        currString = input.substring(currPos, pos);
        commands[count++] = currString;
        currPos = pos+1;
      } else
        break;
```

```
    }
  }
  return count;
}


void loop() {

  // Check UI override
  String commands[20];
  int numCommands = readCommand(commands);
  if (numCommands > 0)
      processCommand(commands, numCommands);

  // Check individual motors
  if (sensorMode) {

      // DC Motor
      updateState(1, 0, 0, 0);
      driveDCMotor();

      // Stepper Motor
      updateStepperState(0, 200, 1);
      updateIRSensorState(1);
      IRSensorAndStepper();

      // Enable Ultrasonic and Servo Motor
      updateUSSensorState(1);
      driveServo();
  } else {
    DC_Motor_Status dc_status;
    get_Motor_Status(dc_status);
    //Serial.println(dc_status.state);
    driveDCMotor();
  }
  writeStatus();
  delay(50);
}



/*
 *  TEST CASES
 *
 * bom,sensor-based,eom
 * bom,m_ser,1,0,na,na,eom
 * bom,m_stp,1,0,na,na,eom
 * bom,m_dcm,1,0,na,0,eom
 * bom,s_usr,1,0,na,na,eom
 * bom,s_isr,1,0,na,na,eom
```

```
* bom,s_for,1,0,na,na,eom
* bom,m_ser,0,0,na,na,eom
* bom,m_ser,1,0,na,na,eom
* bom,m_stp,0,0,na,na,eom
* bom,m_stp,1,0,na,na,eom
* bom,m_dcm,0,0,na,na,eom
* bom,m_dcm,1,0,na,na,eom
* bom,s_usr,0,0,na,na,eom
* bom,s_usr,1,0,na,na,eom
* bom,s_irs,0,0,na,na,eom
* bom,s_irs,1,0,na,na,eom
* bom,s_for,0,0,na,na,eom
* bom,s_for,1,0,na,na,eom
* bom,s_for,0,0,na,na,eom
* bom,gui-based,eom
* bom,m_ser,1,0,na,na,eom
* bom,m_stp,1,0,na,na,eom
* bom,m_dcm,1,0,na,0,eom
* bom,s_usr,1,0,na,na,eom
* bom,s_isr,1,0,na,na,eom
* bom,s_for,1,0,na,na,eom
* bom,m_ser,1,20,0,na,eom
* bom,m_ser,1,20,1,na,eom
* bom,m_stp,1,30,0,na,eom
* bom,m_stp,1,30,1,na,eom
* bom,m_dcm,1,na,0,400,eom
* bom,m_dcm,1,na,1,400,eom
* bom,m_dcm,1,39,1,na,eom
* bom,m_dcm,1,39,0,na,eom
*/
```