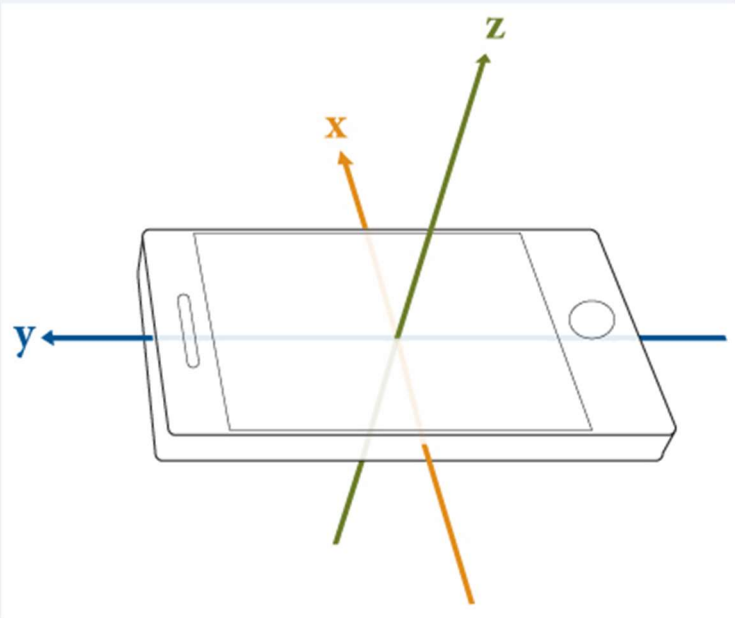
Wireless Controlled Robot Arm Using Mobile Device Sensors



College of Engineering

ECE 5620: Embedded System Design
Fall 2017

Adviser: Dr. Syed Mahmud

Group 7

Rohit Raibagkar
gd4139

Mohammad Mahmoudi Zarandi
fx2618

S M Fairus Hasan
fz8484

Declaration : The project and the report are our own work
Date : 12/19/2017

Abstract

We address the importance of wireless communication protocols in data acquisition systems. Wi-Fi communication is most widely and commonly used wireless communication method. Wi-Fi is based on IEEE 802.11 standards. Considering the speed of communication and data transfer wi-fi is offering, use of wi-fi is not only limited to internet communication but also it can be used for other communications. Wi-Fi is having two communication types, ad-hoc and direct. The wi-fi communication can be used to send data to communicate with hardware such as robot. Since robot needs only one data, set of joint angles, we used wi-fi communication to control joint angles of robot. In the project, we imported the yaw-pitch-roll angle data in computer and parsed the data to robot.

Robotic arms are usually controlled in two limited options. First, the robot is coded with a specific task that is continuously carried out without variation. Second, some sort of live input is used for real-time control of the robotic arm. Various methods are employed to meet such a goal, of which the most popular is using joysticks, for example the robotic arms used for explosive disposal. Using two joysticks to control a robotic arm of usually in 6 degrees of freedom can be rather unintuitive. It can take a long time to train and excel in manipulating the robotic arm for position and orientation using joysticks. Designing a robotic arm that can mimic and move accordingly with the movement of human hand sound promising in the control feasibility of such a robotic arm. Using hand gestures and mapping such motions onto a robotic arm will provide the user with more intuitive control and reduce the learning curve for arm manipulation.

The report is discussion of complete procedure to connect the mobile device to computer, get the sensor data, and controlling the robot arm using the data.

Table of Contents

1.	Introduction	5
2.	Problem Statement	5
3.	Background Materials, Research and Patents	6
4.	Executive Summary	7
5.	Mobile Device Sensors	8
6.	Sensors Data Streaming	9
7.	Connecting Mobile Device to Computer	11
8.	Interfacing of Flex and Servos to Arduino	12
9.	Control of Robot Arm	14
10.	Overcoming Challenges	16
11.	Future Developments	17
12.	Conclusion	18
13.	References	19
14.	Appendix.	
	A. Alternate Design Ideas	20
	B. Constraints due to FCC Regulations	21
	C. Parts List and Cost Analysis	22
	D. Problems	23
	E. Distribution of Work	24
	F. Program Listing	25
	G. Data Sheets	42
	H. YouTube Video Link	43
	I. Program Algorithm Flowchart	44

Table of Figures

Fig. 1	Acceleration and Magnetic Field Sensors	8
Fig. 2	Orientation and Angular Velocity Sensors	8
Fig. 3	Position(GPS) Sensor	8
Fig. 4	Flex Sensor operation: Varying the resistance by bending	12
Fig. 5	Circuit diagram of flex sensor output voltage measurement	12
Fig. 6	Circuit diagram of Flex and Servo Interfacing	13
Fig. 7	Complete Block Diagram of communication and control	15
Fig. 8	GUI of Remote Controlled Robot	41
Fig. 9	Complete assembly of robot and Sensors	41
Fig. 10	Side view of Design	41
Fig. 11	Top view of Design	41

1. Introduction

Wi-fi is IEEE 802.11 standard protocol. It is most commonly used for wireless communication between ethernet devices. Robot is dynamic device which carries out assigned task automatically. Considering the dynamics involved in robot operations, some reliable wireless communication is must to avoid complications and losses involved in wired communications.

A robotic arm is programmable electromechanical system with similar functions to a human arm. The links of the manipulator are connected by joints allowing rotational motion or translational displacement. The links are considered to form a kinematic chain and terminus of this chain is called the end effector. Robots can be autonomous or semi-autonomous. An autonomous robot is operating on its own based on the data it gets from its sensors and the program and code implemented to it for a specific task and is not directly controlled by human. Most of the industrial robots are required to operate with high speed and high accuracy; hence, are autonomous. But some applications require semi-autonomous or human controlled robots.

Most commonly used robots of human controlled are motion controlled, tactile or touch controlled, and voice recognition controlled systems. The frequently motion control robot is a hand gesture controlled robot that is implemented and developed in this project using the built-in sensors in an iPhone x and Arduino mega. Instead of using a remote control with buttons or joystick, the gesture of the hand is used to control the arm.

This project is based on both wired and wireless communication, where data from the hand gesture sensed with the sensors in a cell phone and transmitted to the laptop over Wi-fi network and are processed through MATLAB software. The processed data and required commands are then transferred through serial communication the Arduino board and eventually implemented to the arm.

2. Problem Statement

The approach towards development of the project is below.

- **Programming** `mobileSensorDataGetter` **class in MATLAB**

The class consists of function to connect the mobile device to computer and log the sensor data values at appropriate sampling rate. The connection is established by user input passwords and manually entering DNS IP Address.

- **Designing and programming MATLAB GUI to operate and visualize robot.**

The GUI consists of easy access and user-friendly buttons and other important communication details such as DNS IP address of computer, password entered by the user and mobile device sensor data.

- **Assembling and Wiring.**

Next important task is assembling and connecting the servos and flex sensors to the Arduino. The assembling also involves mechanical construction of robot.

- **Troubleshooting, debugging and testing.**

The assembled hardware should be tested for its robust and reliable operation. Also, electronic connections should be checked for its correctness in order to avoid short circuits in electrical connections.

- **Reach envelop estimation and mapping.**

Although sensors output 0^0 to 180^0 angles, we cannot write those joint angles to servo directly due reach constraints of the robot. Hence, mapping the values of accelerometer and flex sensor to input servo is necessary.

3. Background Materials, Research and Patents

Wireless communication refers to exchange of information between two nodes is a distance without connecting them to each other physically. Using electromagnetic waves such as satellite, infrared, and radio frequencies the information is transmitted through the air, without any need to wiring. Wireless technologies have different types such as Zigbee, Bluetooth, Wifi…etc.

Wifi is in our scope of interest since we are using it in our project. Wifi is a technology for wireless local area networking with devices based on the IEEE 802.11 standards. When devices talk directly to each other without the need to first talk to an access point it is said that the nodes are operating in ad-hoc mode, aka base station.

A robotic arm is a programmable electromechanical arm with similar functionality to a human arm. There are different robot arms defined based on the functionality of and the joint connection, such as cartesian, cylindrical, spherical, scara…etc. Each of them is efficient for a specific field of functionality.

4. Executive Summary

In contemporary societies, the demand for systems capable of connecting devices remotely for data exchange has been significantly increased, specifically with the ever increasing usage of wireless applications. This project proposes the wireless development of a robotic arm using mobile device built-in sensors and a wired flex sensor. A robotic arm that is capable of pick and place operation and is maneuvered by a cell phone through a wifi connection. It is a three link arm which has a stationary base and can yaw, pitch, and roll inside its reach envelope.

In this project we have established the wireless communication between the mobile device and the base station, and a serial communication between the base station and the actual robotic arm. The orientation of the mobile device is sensed with the accelerometer sensor. Data are processed and showed in the MATLAB mobile app. The application is connected through the wifi with the base station, which is the lap top, and sends the data to the MATLAB. A GUI application is designed in MATLAB environment to manipulate the connection of mobile device and the base station, illustrate and process the received data and map the orientation to the desired robotic arm orientations. The base station is connected to the Arduino through serial port and then to the robot arm.

5. Mobile Device Sensors

MATLAB Mobile device has 5 sensors: Acceleration, Magnetic Field Sensor, Orientation Sensor, Angular Velocity Sensor, GPS(Position) Sensor.

1. Acceleration and Magnetic Field Sensor

The acceleration sensor gives real acceleration of device in X, Y and Z direction in m/s^2. The data may have positive and negative values. Magnetic Field Sensor gives magnetic field around X, Y and Z axis in µT.

2. Orientation and Angular Velocity Sensor

The orientation sensor gives rotation around X, Y and Z axis in degrees. Angular Velocity sensor gives angular velocity around X, Y and Z axis is radian/s.

3. Position(GPS) Sensor

Position(GPS) Sensor gives Latitude and Longitude in degrees. Along with that, it also provides information regarding speed of mobile device, course, altitude and horizontal accuracy.

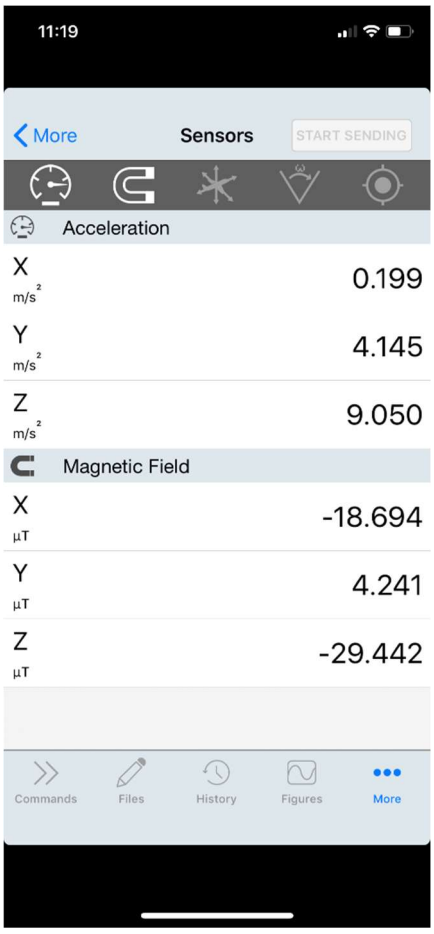


Fig. 1: Acceleration and Magnetic Field Sensors

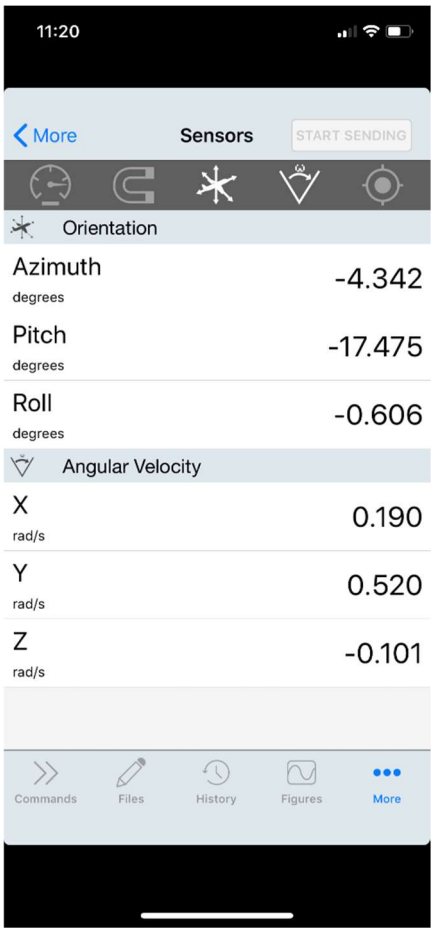


Fig. 2: Orientation and Angular Velocity Sensors

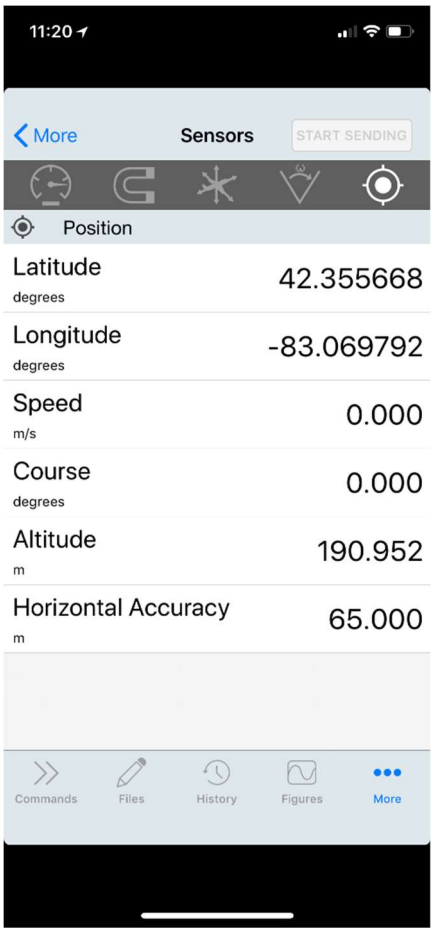


Fig. 3: Position(GPS) Sensor

Out of the five available sensors, we have chosen Orientation sensor to control yaw-pitch-roll of the robot.

6. Sensor Data Streaming

6.1 Create `mobiledev` object

To communicate with iPhone and acquire data streaming from the sensors, we have created `mobiledev` object using `mobiledev` function before turning on the sensors. We cannot get the data without creating object. Also, we must be connected to the same network stream before exchanging the data.

1) We started MATLAB mobile on our iOS device.
2) On the sensors screen of MATLAB Mobile, we turned ON sensors we want to use by tapping the sensors.
3) From our computer running MATLAB, we connected device using `connector` function in MATLAB.

```
connector('on', 'password', password{1,1});
```

Here, we are setting the password to make sure that we are connected to right mobile device. The password is input by user to MATLAB.

4) In MATLAB we created `mobiledev` object in the property of `mobileSensorDataGetter` class.

```
mobile.mobileDevice = mobiledev;     % the property where mobiledevice will
be stored...
```

```
>> mobile.mobileDevice = mobiledev

mobile.mobileDevice =

mobiledev with properties:

                    Connected: 1
                      Logging: 0
             InitialTimestamp: ''

   AccelerationSensorEnabled: 1
AngularVelocitySensorEnabled: 1
       MagneticSensorEnabled: 1
    OrientationSensorEnabled: 1
       PositionSensorEnabled: 1

Supported functions
>>
```

In this display, value of 1 means enabled or ON or true and 0 means disabled or OFF or false. You can see that the device and computer are connected, but data is not being logged yet. This device contains all five sensors, but your device may not. If your device does not have a particular sensor, that sensor will always show a 0 in the display. The timestamp is empty because no data has been logged yet.

5) We began data logging of the sensor by enabling the logging property.

```
mobile.mobileDevice.Logging = 1
```

This action starts transmitting all the data from all the sensors.

6) Now we can display the logging of connected object and sensor data.

```
disp(mobile.mobileDevice)
```

```
mobile.mobileDevice with properties:

                  Connected: 1
                    Logging: 1
             InitialTimestamp: '12-18-2017 13:45:56.529'

   AccelerationSensorEnabled: 1
AngularVelocitySensorEnabled: 1
  MagneticFieldSensorEnabled: 1
    OrientationSensorEnabled: 1
       PositionSensorEnabled: 1

Current Sensor Values:
               Acceleration: [0.27 0.23 -10.19]  (m/s^2)
             AngularVelocity: [-0.22 0.07 0.06]   (rad/s)
               MagneticField: [3.56 1.56 -48.19]  (microtesla)
                 Orientation: [85.91 -27.1 0.35]  (degrees)

       Position Data:
                   Latitude: 41.29  (degrees)
                  Longitude: -72.35  (degrees)
                      Speed: 25  (m/s)
                     Course: 83.6  (degrees)
                   Altitude: 200.1  (m)
          HorizontalAccuracy: 9.0  (m)

Supported functions
```

In the display we can see that now, the device and computer are connected and data is now logged on. The initial time stamp property value displays the indication of current measurement value when object is created.

6.2 Control Sensors and Acquire Data

While we are logging data, we can display the current value of any sensor using the sensor reading properties. The `Acceleration`, `AngularVelocity`, `Orientation`, and `MagneticField` properties display the current readings from their respective sensors. If the Position sensor is logging, we can get individual position readings using `Latitude`, `Longitude`, `HorizontalAccuracy`, `Altitude`, `Course`, and `Speed` properties.

To get the current value from a sensor, use `<objectname>.<propertyname>`.

```
mobile.mobileDevice.Acceleration
ans =

    0.6945   -0.2579    9.9338
```

To get the longitude reading from the Position sensor:

```
m.Longitude
ans =

    -71.3517
```

7. Connecting Mobile Device to Computer

The steps below show how to connect mobile device to computer and assume that mobile device and computer are connected to the same computer.

1) We used MATLAB Connector to set up the connection between our computer running MATLAB and the MATLAB Mobile application on our iOS device. It is mandatory that our computer and device are on the same wireless network. We used following command in MATLAB:

   ```
   connector('on', 'password', 'mypassword');
   ```

   In the password argument, we entered our password.

2) In the MATLAB Mobile application on our device, we selected the option **Connect to Your Computer**.

3) In the **Computer** setting, we entered the DNS name or IP address that was displayed.

4) In the **Connector Password** and **Port** settings, enter the password and port that you previously specified.

5) Saved our settings and started the connection by tapping the **Connect** button at the top of the screen.

The sensors buttons display the following measurements:

- **Acceleration** — the three data points are the acceleration reading in X, Y, and Z coordinates, in m/s^2 (meters per second squared).
- **Magnetic Field** — the three data points are the magnetic field reading in X, Y, and Z coordinates, in microtesla.
- **Orientation** — the three data points are the position reading in X, Y, and Z coordinates, in degrees, for azimuth, pitch, and roll.
- **Angular Velocity** — the three data points are the angular velocity reading in X, Y, and Z rotations, in radians per second.
- **Position** — data points representing latitude, longitude, speed, course, altitude, and horizontal accuracy.

8. Interfacing of Flex and Servos to Arduino

8.1 Interfacing Flex Sensor to Arduino

Flex sensor is flexible resistor which gives variable output voltage. He resistance is varied by bending the flex as shown in the figure below. Its flat resistance is 7 to 13 kOhms. When it is bent 180^0 its resistance becomes 2X of flat resistance. This varied resistance gives output voltage. This output voltage is given to analog input pin 'A0' of the Arduino.

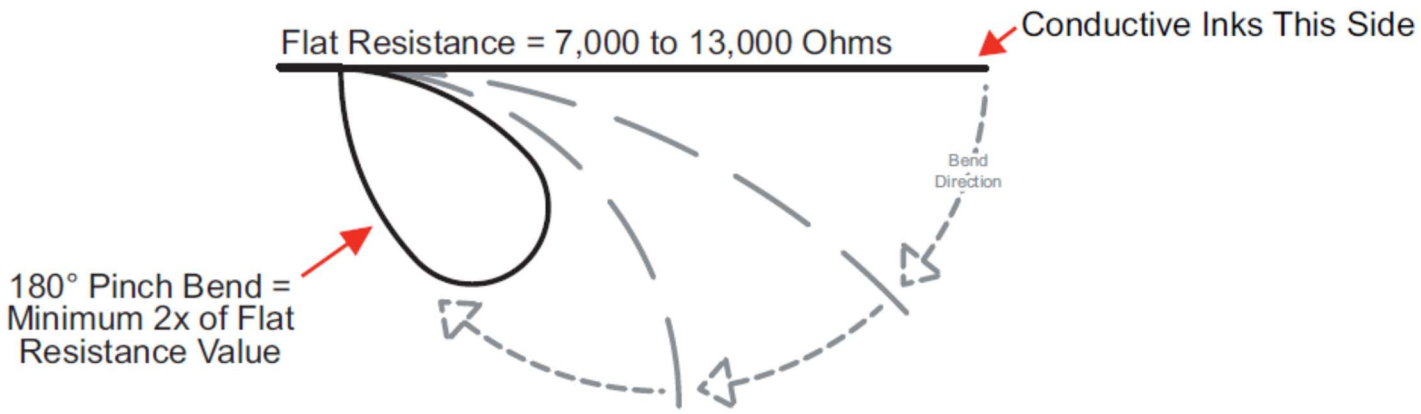


Fig. 4: Flex Sensor operation: Varying the resistance by bending.

Figure below describes the circuit diagram of flex sensor to interface with Arduino. We are not using any op-amp as input buffer. The R2 we are using is having resistance of 18kOhms.

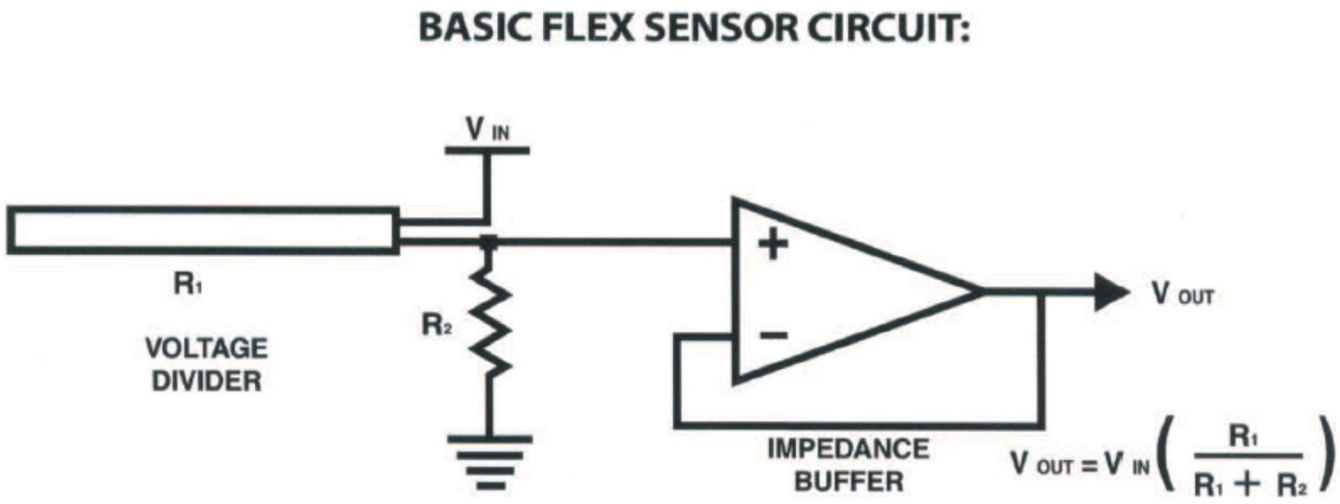


Fig. 5: Circuit diagram of flex sensor output voltage measurement.

8.2 Interfacing Servos to Arduino

The Servos to actuate the robot are controlled by Arduino and the data controlling the servos comes from either the accelerometer sensor or flex sensor. We built the robot and calculated the reach envelop of the robot. The table below shows the role of each servo and sensor value controlling it.

Servo	Actuation Purpose	Sensor Controlling	Working Range (Mapped Angle)	Pin Number
Servo 1	Yaw (z-axis rotation) control	Mobile Device Accelerometer, Yaw value	45^0 to 135^0	D2
Servo 2	Pitch (y-axis rotation) control	Mobile Device Accelerometer, Pitch value	53^0 to 126^0	D3
Servo 3	Roll (x-axis rotation) control	Mobile Device Accelerometer, Roll value	45^0 to 135^0 Center: 90^0	D4
Servo 4	Clutching and Declutching	Flex Sensor	13.5^0 to 36^0	D5

The servos we use are HS-53 Analog sub-micro servo. They operate on 4.8 V. The servos are controlled by digital pin of Arduino. The complete circuit diagram is as shown below.

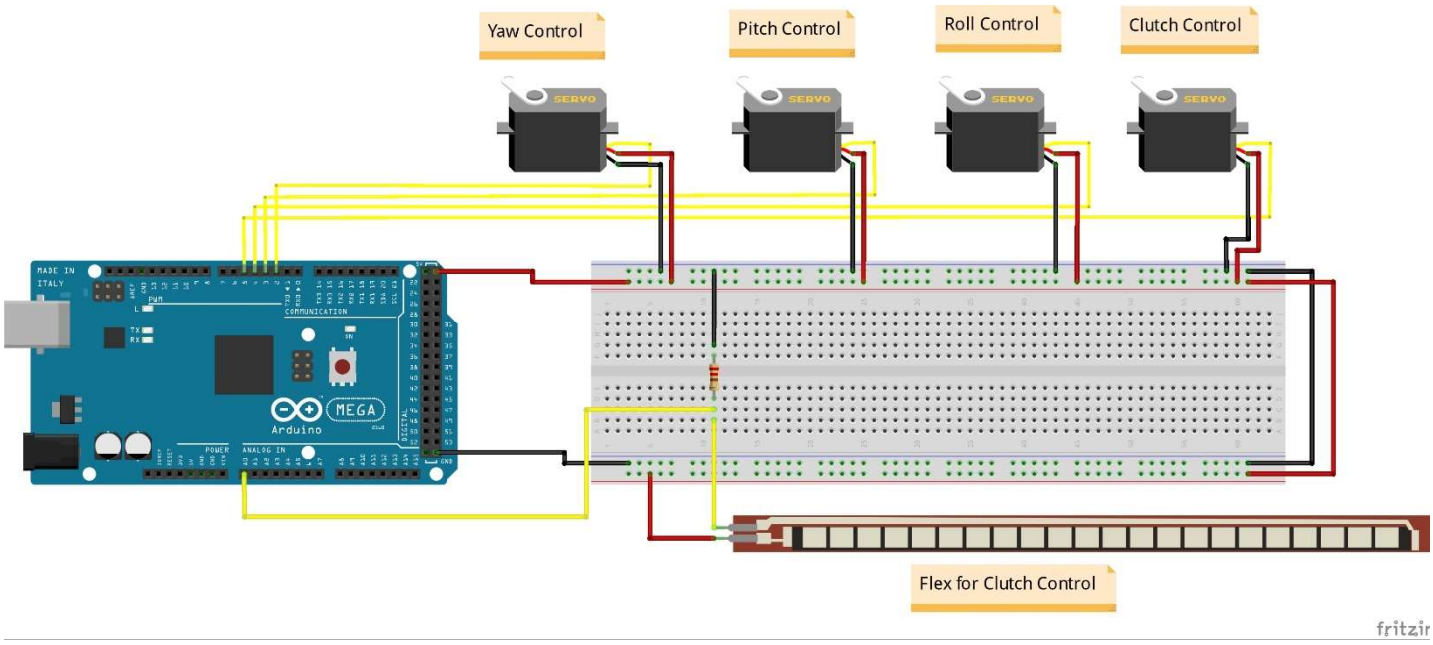


Fig. 6: Circuit diagram of Flex and Servo Interfacing

9. Control of Robot Arm

The Robot arm is controlled by the joint angle data obtained from accelerometer of mobile device. The sensor data is written to serial port using MATLAB. Then Arduino takes those joint angles in and writes them to the servo.

The program below shows the sequence of control of robotic arm.

```
try
    while handles.mobileBotController.mobileDevice.Connected == true
&& handles.mobileBotController.mobileDevice.Logging == true...
&& handles.mobileBotController.mobileDevice.AccelerationSensorEnabled == true...
&& handles.mobileBotController.mobileDevice.AngularVelocitySensorEnabled ==
true...
&& handles.mobileBotController.mobileDevice.MagneticSensorEnabled == true...
&& handles.mobileBotController.mobileDevice.OrientationSensorEnabled == true...
&& handles.mobileBotController.mobileDevice.PositionSensorEnabled == true;

        if handles.stop_bit == true;
            handles.stop_bit = false;
            break;
        elseif handles.stop_bit == 0;
            set(handles.sensor_data_log_display, 'String',
matlab.unittest.diagnostics.ConstraintDiagnostic.getDisplayableString(handles.mob
ileBotController.mobileDevice));
            jointAngles = handles.mobileBotController.mobileDevice.Orientation;

            flexVoltage = (readVoltage(handles.mobileBotController.a, 'A0')/5) -
0.1;
            if jointAngles(1,1)/180 >=0.25 && jointAngles(1,1)/180 <= 0.75;
                writePosition(handles.mobileBotController.s1,
jointAngles(1,1)/180); % yaw
            end

            if jointAngles(1,2)/60 >= 0.2 && jointAngles(1,2)/60 <= 0.7;
                writePosition(handles.mobileBotController.s2,
jointAngles(1,2)/60); % pitch
            end

            if jointAngles(1,3)/180 >= 0.25 && jointAngles(1,3)/180 <= 0.75;
                writePosition(handles.mobileBotController.s3,
jointAngles(1,3)/180); % roll
            end

            if flexVoltage <=0.075
                writePosition(handles.mobileBotController.s4, 0.075); % clutch
            elseif flexVoltage >= 0.2
                writePosition(handles.mobileBotController.s4, 0.2); % declutch
            end

            set(handles.joint_angles, 'String', sprintf('Joint angles : [%f %f
%f]', jointAngles(1,1), jointAngles(1,2), jointAngles(1,3)));
            set(handles.flex_value, 'String', sprintf('Clutch Value : %f',
flexVoltage));

        end
```

```
        pause(.075);
        guidata(hObject, handles);
    end
catch
    errordlg('Please enable mobile device connector, connect mobile device and
enable data logging', 'Device not connected !!', 'modal');
end
guidata(hObject, handles);
```

First, the joint angle of servo 1 are mapped to appropriate value and written to servo 1. Then same procedure is followed for consequent servos. The loop first checks for the conditions of mobile device connection, Data logging status, sensors enabled. If all the conditions are satisfied and true, then consequent operations are carried out.

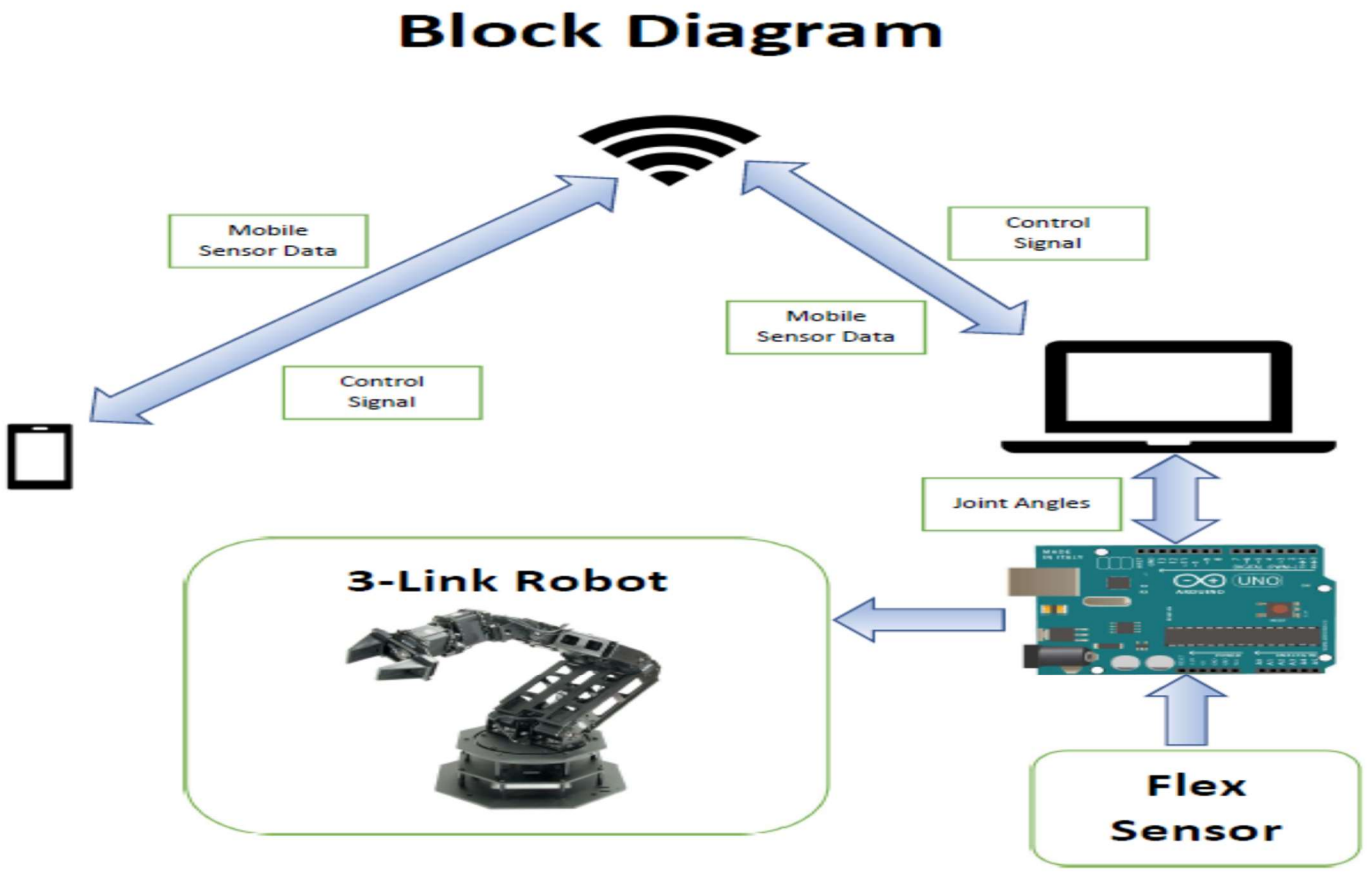


Fig. 7: Complete Block Diagram of communication and control

10. Overcoming Challenges

We faced few technical problems while designing and developing robot as well as while implementing computer program. The major problems are discussed here.

1. **Slow Data value(Latency)**

   The data value or number of logged data value depends upon the sampling rate of data logging. MATLAB Mobile device has three options for sampling rate: `min`, `medium` and `max`. `samplerate` Property of the mobile device sensor sets the sampling rate of the mobile device data logging in hertz. The value is of type `double` and must be in the range of 0 Hz to 100 Hz. The default value is 10 Hz. We can change the value before or after starting the logging of data.

   ```
   mobile.mobileDevice.SampleRate = str2double(inputdlg({'Enter Sampling Rate
   from min 1 to max 100'}, 'Enter Sampleing Rate', [1 50], {'10'}, options));
   ```

   The command takes appropriate sampling value as input from user and then converts it to double. We used 100 Hz (max) as our sampling rate and we got good and improved data logging speed.

2. **Maximum value of pitch to 90^0**

   The maximum pitch value that accelerometer can offer is 90^0. Beyond that its data value starts decreasing. Yaw and roll values range from 0^0 to 180^0. But only pitch value is having maximum range of 90^0. Hence we had to map the values in between 0^0 to 90^0 using suitable scaling factor.

3. **Loss of Wi-Fi Connection**

   Wi-Fi connection sometimes becomes slow or completely lost. When we lose connection, we have to restart the connection procedure to MATLAB mobile device. Hence dedicated wi-fi connection is suggested for communication. Also internet traffic on the Wi-fi node affects the data transfer speed. Hence, use of dedicated Wi-Fi node is recommended.

11. Future Developments

Future developments include Use of Bluetooth shield, Voice recognition(commands), Addition of Camera, Using accelerometer to track x-y-z coordinates of object in 3-D space.

1. Use of Bluetooth Shield

Bluetooth shield of Arduino provides uninterrupted wireless communication. But the cost of Bluetooth communication is same as new Wi-Fi router. Hence Detailed research is necessary for Bluetooth compatibility. But Bluetooth can be used to transmit data from flex sensor to main Arduino.

2. Voice Recognition

Voice recognition can be added to make robot move in the direction user want. Instead of clicking each button in MATLAB, we can use voice commands in MATLAB to operate the robot.

3. Addition of Camera for Object Detection

Camera can be used for the object and pattern detection. The camera will be streaming images of the object that is in the vicinity of the robot. We can make the robot to choose the predefined object and pick and place to the desired location. The camera will be either controlled by computer vision or raspberry pi controller.

4. Track x-y-z coordinates using accelerometer

The accelerometer can be used to track the x-y-z coordinates of the object in 3D space. Using the co-ordinates, we can make robot to move to the desired location in 3D space. The accelerometer data will be streamed using Wi-Fi Connection.

5. Use of Parallel Systems to Write the Joint Angles

We can use parallel pool in MATLAB or parallel computing in the embedder systems to parse the joint angles simultaneously. This will be complete utilization of the computer memory and will also increase the speed of robot operation.

12. Conclusion and Results

Hence we successfully completed the project. We planned our project as mentioned in the problem statement. Then we managed to connect the mobile device sensors to computer as discussed in Mobile Device Sensors Section. We managed to acquire sensor data as discussed in sensor data streaming. We built robot arm and controlled it through acquired sensor data. We also encountered some problems and technical challenges. We explored through MATLAB to learn about data acquisition. We successfully implemented the task using

- Arduino Mega 2560 : To control servo and get data from flex sensors.
- Flex : to sense the bending of fingers for clutching and declutching.
- Servo: to actuate the joint angle of robot arm.

Implementing this technology in robotic operations will bring ease of control of robot with remote access.

13. References

Refrences

A. Sarkar, K. A. Patel, R. K. G. Ram, and G. K. Capoor, “Gesture control of drone using a motion controller,” in 2016 International Conference on Industrial Informatics and Computer Systems (CIICS), March 2016, pp. 1–5.

G. C. Luh, H. A. Lin, Y. H. Ma, and C. J. Yen, “Intuitive musclegesture based robot navigation control using wearable gesture armband,” in Machine Learning and Cybernetics (ICMLC), 2015 International Conference on, vol. 1, July 2015, pp. 389–395.

J. L. Raheja, R. Shyam, U. Kumar, and P. B. Prasad, “Real-time robotic hand control using hand gestures,” in Machine Learning and Computing (ICMLC), 2010 Second International Conference on, Feb 2010, pp. 12– 16.

G. C. B and C. R. B. V, “Real time robotic arm control using hand gestures,” in High Performance Computing and Applications (ICHPCA), 2014 International Conference on, Dec 2014, pp. 1–3.

F. Kobayashi, K. Okamoto, and F. Kojima, “Robot-human handover based on position and posture of human hand,” in Soft Computing and Intelligent Systems (SCIS), 2014 Joint 7th International Conference on and Advanced Intelligent Systems (ISIS), 15th International Symposium on, Dec 2014, pp. 918–921.

www.electronicshub.org/hand-gesture-controlled-robot/

https://learn.sparkfun.com/

14. Appendix

A. Alternate Design Ideas

Future developments include Use of Bluetooth shield, Voice recognition(commands), Addition of Camera, Using accelerometer to track x-y-z coordinates of object in 3-D space.

1. Use of Bluetooth Shield

Bluetooth shield of Arduino provides uninterrupted wireless communication. But the cost of Bluetooth communication is same as new Wi-Fi router. Hence Detailed research is necessary for Bluetooth compatibility. But Bluetooth can be used to transmit data from flex sensor to main Arduino.

2. Voice Recognition

Voice recognition can be added to make robot move in the direction user want. Instead of clicking each button in MATLAB, we can use voice commands in MATLAB to operate the robot.

3. Addition of Camera for Object Detection

Camera can be used for the object and pattern detection. The camera will be streaming images of the object that is in the vicinity of the robot. We can make the robot to choose the predefined object and pick and place to the desired location. The camera will be either controlled by computer vision or raspberry pi controller.

4. Track x-y-z coordinates using accelerometer

The accelerometer can be used to track the x-y-z coordinates of the object in 3D space. Using the co-ordinates, we can make robot to move to the desired location in 3D space. The accelerometer data will be streamed using Wi-Fi Connection.

5. Use of Parallel Systems to Write the Joint Angles

We can use parallel pool in MATLAB or parallel computing in the embedder systems to parse the joint angles simultaneously. This will be complete utilization of the computer memory and will also increase the speed of robot operation.

B. Constraints due to FCC Regulations

The Federal Communications Commission (FCC) has which regulate power for Wi-Fi networks. From the viewpoint of the FCC regulations, the power of a Wi-Fi broadcast is measured in units of equivalent isotropically radiated power (EIRP). EIRP represents the total effective transmitting power of the radio in a Wi-Fi card or access point, including adding gains from an antenna and subtracting losses from an antenna cable. When using an omnidirectional antenna (see "Different Kinds of Antennas" earlier in this chapter for more information about types of antennas) with fewer than 6 decibels (dB) gain, the FCC requires EIRP to be under one watt.

- antenna gain counts
- cable signal loss counts
- One Watt (1000mw) is the FCC limit on WiFi devices

You cannot get a more powerful wireless device without an FCC license than 1 watt for 802.11x protocols.

The FCC Regs say:

```
(b) The maximum peak output power of the intentional radiator
shall not exceed the following:
(1) For frequency hopping systems in the 2400-2483.5 MHz band
employing at least 75 hopping channels, and all frequency hopping
systems in the 5725-5850 MHz band: 1 Watt. For all other frequency
hopping systems in the 2400-2483.5 band: 0.125 Watt.

(4) Except as shown in paragraphs (b)(3) (i), (ii) and (iii) of this
section, if transmitting antennas of directional gain greater than 6 dBi
are used the peak output power from the intentional radiator shall be
reduced below the stated values in paragraphs (b)(1) or (b)(2) of this
section, as appropriate, by the amount in dB that the directional gain
of the antenna exceeds 6 dBi.
```

Which may indicate with a high gain antenna, up to 4 watts is permitted. Furthermore, there seems to be an exemption for point to point communications.

```
Sec. 15.247 Operation within the bands 902-928 MHz, 2400-2483.5 MHz, and 5725-5850 MHz.

(i) Systems operating in the 2400-2483.5 MHz band that are used
exclusively for fixed, point-to-point operations may employ transmitting
antennas with directional gain greater than 6 dBi provided the maximum
peak output power of the intentional radiator is reduced by 1 dB for
every 3 dB that the directional gain of the antenna exceeds 6 dBi.
```

Source: http://wiki.robotz.com/index.php/FCC_Regulations_on_WiFi

C. Parts List and Cost Analysis

Part	Quantity	Price
Servo	4	$ 27
3D Printed Parts	Not Applicable	$ 25
Arduino Mega 2560	1	$ 32
Flex Sensors	2	$ 14
Breadboard	1	$ 5
Glove	1	$ 7
Connecting Wires	1 Set	$ 6
Screws, Nuts and Bolts	1 Pack	$ 11
	Total	**$127**

D. Problems

We faced few technical problems while designing and developing robot as well as while implementing computer program. The major problems are discussed here.

1. **Slow Data value(Latency)**

   The data value or number of logged data value depends upon the sampling rate of data logging. MATLAB Mobile device has three options for sampling rate: `min`, `medium` and `max`. `samplerate` Property of the mobile device sensor sets the sampling rate of the mobile device data logging in hertz. The value is of type `double` and must be in the range of 0 Hz to 100 Hz. The default value is 10 Hz. We can change the value before or after starting the logging of data.

```
mobile.mobileDevice.SampleRate = str2double(inputdlg({'Enter Sampling Rate
from min 1 to max 100'}, 'Enter Sampleing Rate', [1 50], {'10'}, options));
```

   The command takes appropriate sampling value as input from user and then converts it to double. We used 100 Hz (max) as our sampling rate and we got good and improved data logging speed.

2. **Maximum value of pitch to 90^0**

   The maximum pitch value that accelerometer can offer is 90^0. Beyond that its data value starts decreasing. Yaw and roll values range from 0^0 to 180^0. But only pitch value is having maximum range of 90^0. Hence we had to map the values in between 0^0 to 90^0 using suitable scaling factor.

3. **Loss of Wi-Fi Connection**

   Wi-Fi connection sometimes becomes slow or completely lost. When we lose connection, we have to restart the connection procedure to MATLAB mobile device. Hence dedicated wi-fi connection is suggested for communication. Also internet traffic on the Wi-fi node affects the data transfer speed. Hence, use of dedicated Wi-Fi node is recommended.

E. Distribution of Work

	Rohit	Mohammad	Hasan
Brainstorming Ideas	33.33 %	33.33 %	33.33 %
Research	33.33 %	33.33 %	33.33 %
Flex sensor assemble	10 %	10 %	80 %
Design	25 %	25 %	50 %
Connections	15 %	15 %	70 %
Programming	80 %	10 %	10 %
GUI Programming	25 %	70 %	5 %
Documentation	40 %	40 %	20 %
Prototype Assembly	33.33 %	33.33 %	33.33 %
Presentation	33.33 %	33.33 %	33.33 %
Troubleshooting	33.33 %	33.33 %	33.33 %

F. Program Listing

1. `mobileSensorDataGetter` Class Code

```
classdef mobileSensorDataGetter
    %UNTITLED2 Summary of this class goes here
    %   Detailed explanation goes here

    properties

        mobileDevice            % The variable where mobile device class is
stored...
        connectionSummary       % The variable where details of the connection
are stored...
        loggingStatus           % The variable where status of logging is
stored...

        accelerometerStatus     % Status of accelerometer...
        gyroSensorStatus        % Status of gyro Sensor...
        magnetoStatus           % Status of magnetic sensor...
        orientation             % Status of orientation sensor...
        position                % Status of position sensor...

        a                       % arduino connection variable...
        s1                      % servo 1: yaw control
        s2                      % servo 2: pitch control
        s3                      % servo 3: roll control
        s4                      % servo 4: clutch and de-clutch control

    end

    methods

        function mobile = connectMyMobileDev (mobile) % Opens port to connect
mobile device to MATLAB...

            options.Resize = 'on';
            options.WindowStyle = 'normal';
            options.Interpreter = 'tex';
            password = inputdlg({'Enter Alphanumeric password containing more
than 5 characters'},...
                                'Enter Alpha-numeric Password', [1 50], {''},
options); % to be displayed
            connector('on', 'password', password{1,1});
            hostname = char(getHostName(java.net.InetAddress.getLocalHost)); % to
be displayed
            ip_address = char(getHostAddress(java.net.InetAddress.getLocalHost));
% to be displayed
            user = getenv('UserName'); % to be displayed
            mobile.connectionSummary = sprintf('Connetion details are : \n\nDNS
name : %s \nIP address : %s \nPassword : %s \nUser : %s',...
                                               hostname, ip_address,
password{1,1}, user); % connecton summary of the mobile device...
```

```
            mobile.mobileDevice = mobiledev;    % the property where mobiledevice
will be stored...
            d = dialog('Position', [400 400 300 190],'Name','Ready to connect
!!');

            txt = uicontrol('Parent',d,...
                'Style','text',...
                'Position', [45 100 210 80],...
                'String',mobile.connectionSummary);

            btn = uicontrol('Parent',d,...
                'Position', [100 60 100 25],...
                'String','Close',...
                'Callback','delete(gcf)');

        end

        function mobile = disconnectMyMobileDev (mobile) % Closes port of Mobile
device connected to MATLAB...

            mobile.mobileDevice = 0; % sets mobile device to zero
            clear mobile.mobileDevice; % clears mobile device
            connector off;
            mobile.connectionSummary = sprintf('Mobile Device successfully
disconnected...!!!');
            msgbox('Mobile Device has been successfully disconnected.',
'Success', 'modal');

        end

        function mobile = enableDataLogging (mobile) % Enables data logging of
mobile device sensors...

            if mobile.mobileDevice.Connected == 0;
                errordlg('Please connect Mobile Device to Matlab', 'Device not
Connected !!', 'modal');
            elseif mobile.mobileDevice.Connected == 1;
                options.Resize = 'on';
                options.WindowStyle = 'normal';
                options.Interpreter = 'tex';
                mobile.mobileDevice.SampleRate = str2double(inputdlg({'Enter
Sampling Rate from min 1 to max 100'}, ...
                    'Enter Sampleing Rate', [1 50], {'10'}, options)); % to be
displayed
                if mobile.mobileDevice.Connected == true &&
(mobile.mobileDevice.AccelerationSensorEnabled == true ...
                        || mobile.mobileDevice.AngularVelocitySensorEnabled ==
true ...
                        || mobile.mobileDevice.MagneticSensorEnabled == true ...
                        || mobile.mobileDevice.OrientationSensorEnabled == true
...
                        || mobile.mobileDevice.PositionSensorEnabled == true);

                    mobile.mobileDevice.Logging = true; % data logging is enabled
only if all the sensors are enebled...
```

```
                    msgbox('Data Logging has been successfully enabled.',
'Success', 'modal');
                    mobile.loggingStatus = sprintf('Data
Logging.................:: Enabled');

                elseif mobile.mobileDevice.Connected == true &&
(mobile.mobileDevice.AccelerationSensorEnabled == false ...
                        && mobile.mobileDevice.AngularVelocitySensorEnabled ==
false ...
                        && mobile.mobileDevice.MagneticSensorEnabled == false ...
                        && mobile.mobileDevice.OrientationSensorEnabled == false
...
                        && mobile.mobileDevice.PositionSensorEnabled == false);
                    errordlg('Please enable at least one Sensor of your Mobile
device to enable data logging',...
                        'Sensor not enabled !!', 'modal');
                end
            end
        end

        function mobile = disableDataLogging (mobile) % disables data logging...

            if mobile.mobileDevice.Connected == 0;
                errordlg('Please connect Mobile device to Matlab', 'Device not
Connected !!', 'modal');
            elseif mobile.mobileDevice.Connected == true &&
mobile.mobileDevice.Logging == false;
                errordlg('Data Logging is already disabled.', 'Logging disabled
!!', 'modal');
            elseif mobile.mobileDevice.Connected == true &&
mobile.mobileDevice.Logging == true
                mobile.mobileDevice.Logging = false;
                msgbox('Data Logging has been successfully disabled.', 'Success',
'modal');
                mobile.loggingStatus = sprintf('Data Logging.................::
Disabled');
            end
        end

        function mobile = enableAccelerometer (mobile) % enables accelerometer...

            if mobile.mobileDevice.Connected == true &&
mobile.mobileDevice.AccelerationSensorEnabled == false;
                mobile.mobileDevice.AccelerationSensorEnabled = true;
                msgbox('Accelerometer has been successfully enabled.', 'Success',
'modal');
                mobile.accelerometerStatus =
sprintf('Accelerometer...............:: Enabled');
            elseif mobile.mobileDevice.Connected == true &&
mobile.mobileDevice.AccelerationSensorEnabled == true;
                msgbox('Accelerometer has already been enabled...!!', 'Status',
'modal');
            end

        end
```

```
        function mobile = enableGyroSensor (mobile) % enables gyro sensor...

            if mobile.mobileDevice.Connected == true &&
mobile.mobileDevice.AngularVelocitySensorEnabled == false;
                mobile.mobileDevice.AngularVelocitySensorEnabled = true;
                msgbox('Gyro sensor has been successfully enabled.', 'Success',
'modal');
                mobile.gyroSensorStatus = sprintf('Gyro Sensor.................::
Enabled');
            elseif mobile.mobileDevice.Connected == true &&
mobile.mobileDevice.AngularVelocitySensorEnabled == true;
                msgbox('Gyro Sensor has already been enabled...!!', 'Status',
'modal');
            end

        end

        function mobile = enableMagneticSensor (mobile) % enables magnetic
sensor...

            if mobile.mobileDevice.Connected == true &&
mobile.mobileDevice.MagneticSensorEnabled == false;
                mobile.mobileDevice.MagneticSensorEnabled = true;
                msgbox('Magnetic sensor has been successfully enabled.',
'Success', 'modal');
                mobile.magnetoStatus = sprintf('Magnetic Sensor...........::
Enabled');
            elseif mobile.mobileDevice.Connected == true &&
mobile.mobileDevice.MagneticSensorEnabled == true;
                msgbox('Magnetic Sensor has already been enabled...!!', 'Status',
'modal');
            end

        end

        function mobile = enableOrientation (mobile) % enables orientation
sensor...

            if mobile.mobileDevice.Connected == true &&
mobile.mobileDevice.OrientationSensorEnabled == false;
                mobile.mobileDevice.OrientationSensorEnabled = true;
                msgbox('Orientation sensor has been successfully enabled.',
'Success', 'modal');
                mobile.orientation = sprintf('Orientation Sensor........::
Enabled');
            elseif mobile.mobileDevice.Connected == true &&
mobile.mobileDevice.OrientationSensorEnabled == true;
                msgbox('Orientation Sensor has already been enabled...!!',
'Status', 'modal');
            end

        end
```

```
        function mobile = enablePosition (mobile) % this function enables
position sensor

            if mobile.mobileDevice.Connected == true &&
mobile.mobileDevice.PositionSensorEnabled == false;
                mobile.mobileDevice.PositionSensorEnabled = true;
                msgbox('Position sensor (GPS) has been successfully enabled.',
'Success', 'modal');
                mobile.position = sprintf('Position (GPS) Sensor..:: Enabled');
            elseif mobile.mobileDevice.Connected == true &&
mobile.mobileDevice.PositionSensorEnabled == true;
                msgbox('Position Sensor (GPS) has already been enabled...!!',
'Status', 'modal');
            end

        end

        function mobile = enableAllSensors (mobile) % this function enables all
the sensors at once...

            mobile = mobile.enableAccelerometer;
            mobile = mobile.enableGyroSensor;
            mobile = mobile.enableMagneticSensor;
            mobile = mobile.enableOrientation;
            mobile = mobile.enablePosition;

        end

        function mobile = disableMobileSensors (mobile) % the function disables
the mobile device sensors...

            Sensors = {'Accelerometer' 'Gyro Sensor' 'Magnetic Sensor'...
                'Orientation Sensor' 'Position Sensor (GPS)' 'Turn Off all'};
            [s,v] = listdlg('ListString', Sensors, 'SelectionMode', 'single',...
                'ListSize', [173 80], 'InitialValue', [1], 'Name', 'Sensor',...
                'PromptString', 'Please select Sensors you want to Turn Off
:',...
                'OKString', 'Turn Off', 'CancelString', 'Cancel');
            if v == 1;
                switch s
                    case 1 % if first choice is selected...
                        mobile.mobileDevice.AccelerationSensorEnabled = false; %
accelerometer is disabled...
                        disp('Accelerometer Turned Off ...');
                        msgbox('Accelerometer has been successfully disabled.',
'Success', 'modal');
                        mobile.accelerometerStatus =
sprintf('Accelerometer..............:: Disabled');
                    case 2 % if second choice is selected...
                        mobile.mobileDevice.AngularVelocitySensorEnabled = false;
% angular velocity sensor is disabled...
                        disp('Gyro Sensor Turned Off ...');
                        msgbox('Gyro sensor has been successfully disabled.',
'Success', 'modal');
```

```
                        mobile.gyroSensorStatus = sprintf('Gyro
Sensor.................:: Disabled');
                    case 3 % if third choice is selected...
                        mobile.mobileDevice.MagneticSensorEnabled = false; %
magnetic sensor is disabled...
                        disp('Magnetic Sensor Turned Off ...');
                        msgbox('Magnetic sensor has been successfully disabled.',
'Success', 'modal');
                        mobile.magnetoStatus = sprintf('Magnetic
Sensor...........:: Disabled');
                    case 4 % if fourth option is selected...
                        mobile.mobileDevice.OrientationSensorEnabled = false; %
Orientation sensor is disabled...
                        disp('Orientation Sensor Turned Off ...');
                        msgbox('Orientation sensor has been successfully
disabled.', 'Success', 'modal');
                        mobile.orientation = sprintf('Orientation
Sensor........:: Disabled');
                    case 5 % if fifth choice is selected...
                        mobile.mobileDevice.PositionSensorEnabled = false; % gps
sensor is disabled...
                        disp('Position Sensor (GPS) Turned Off ...')
                        msgbox('Position sensor (GPS) has been successfully
disabled.', 'Success', 'modal');
                        mobile.position = sprintf('Position (GPS) Sensor..::
Disabled');
                    case 6 % if sixth choice is selected, all the sensors are
disabled...
                        mobile.mobileDevice.AccelerationSensorEnabled = false;
                        mobile.mobileDevice.AngularVelocitySensorEnabled = false;
                        mobile.mobileDevice.MagneticSensorEnabled = false;
                        mobile.mobileDevice.OrientationSensorEnabled = false;
                        mobile.mobileDevice.PositionSensorEnabled = false;
                        disp('All sensors Turned Off');
                        msgbox('All the sensors has been successfully disabled.',
'Success', 'modal');
                        mobile.accelerometerStatus =
sprintf('Accelerometer..............:: Disabled');
                        mobile.gyroSensorStatus = sprintf('Gyro
Sensor.................:: Disabled');
                        mobile.magnetoStatus = sprintf('Magnetic
Sensor...........:: Disabled');
                        mobile.orientation = sprintf('Orientation
Sensor........:: Disabled');
                        mobile.position = sprintf('Position (GPS) Sensor..::
Disabled');
                        mobile.loggingStatus = sprintf('Data
Logging................:: Disabled');
                    otherwise
                        disp('Please select sensor you want to Turn Off ...')
                end

            elseif v == 0
                disp('Please select sensor you want to Turn Off ...');
            end
```

```
        end

        function mobile = connectArduino (mobile) % this function connects the
arduino to matlab...

            mobile.a = arduino; % stores arduino class to property a

            mobile.s1 = servo(mobile.a, 'D2'); % stores yaw control servo to s1
property and
            writePosition(mobile.s1, .25); % writes initial joint angle to yaw
control servo

            mobile.s2 = servo(mobile.a, 'D3'); % stores pitch control servo to s2
property and
            writePosition(mobile.s2, .3); % writes initial joint angle to pitch
control servo

            mobile.s3 = servo(mobile.a, 'D4'); % stores roll control servo to s3
property
            writePosition(mobile.s3, .5); % writes initial joint angle to pitch
control servo

            mobile.s4 = servo(mobile.a, 'D5'); % stores clutch control servo to
s4 property and
            writePosition(mobile.s4, .075); % writes initial joint angle to
clutch control servo

            configurePin(mobile.a, 'A0');
        end

        function mobile = disConnectArduino (mobile) % the function to disconnect
arduino...

            clear mobile.a;
            mobile.a = 0;
            clear mobile.s1;
            mobile.s1 = 0;
            clear mobile.s2;
            mobile.s2 = 0;
            clear mobile.s3;
            mobile.s3 = 0;
            clear mobile.s4;
            mobile.s4 = 0;

        end

    end

end
```

2. GUI Code

```
function varargout = mobileBotControl_ECE5620(varargin)
% MOBILEBOTCONTROL_ECE5620 MATLAB code for mobileBotControl_ECE5620.fig
%      MOBILEBOTCONTROL_ECE5620, by itself, creates a new
MOBILEBOTCONTROL_ECE5620 or raises the existing
%      singleton*.
%
%      H = MOBILEBOTCONTROL_ECE5620 returns the handle to a new
MOBILEBOTCONTROL_ECE5620 or the handle to
%      the existing singleton*.
%
%      MOBILEBOTCONTROL_ECE5620('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in MOBILEBOTCONTROL_ECE5620.M with the given input
arguments.
%
%      MOBILEBOTCONTROL_ECE5620('Property','Value',...) creates a new
MOBILEBOTCONTROL_ECE5620 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before mobileBotControl_ECE5620_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to mobileBotControl_ECE5620_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help mobileBotControl_ECE5620

% Last Modified by GUIDE v2.5 15-Dec-2017 21:00:19

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @mobileBotControl_ECE5620_OpeningFcn, ...
                   'gui_OutputFcn',  @mobileBotControl_ECE5620_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before mobileBotControl_ECE5620 is made visible.
function mobileBotControl_ECE5620_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to mobileBotControl_ECE5620 (see VARARGIN)

% Choose default command line output for mobileBotControl_ECE5620
handles.output = hObject;
mobileSensorDataGetter;
handles.mobileBotController = mobileSensorDataGetter;
handles.stop_bit = false;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes mobileBotControl_ECE5620 wait for user response (see UIRESUME)
% uiwait(handles.main_axes);


% --- Outputs from this function are returned to the command line.
function varargout = mobileBotControl_ECE5620_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --------------------------------------------------------------------
function mobile_device_Callback(hObject, eventdata, handles)
% hObject    handle to mobile_device (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --------------------------------------------------------------------
function connect_mobile_device_Callback(hObject, eventdata, handles)
% hObject    handle to connect_mobile_device (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.mobileBotController = handles.mobileBotController.connectMyMobileDev;
set(handles.connection_summary, 'String',
handles.mobileBotController.connectionSummary);
guidata(hObject, handles);

% --------------------------------------------------------------------
function disconnect_mobile_device_Callback(hObject, eventdata, handles)
% hObject    handle to disconnect_mobile_device (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.mobileBotController = handles.mobileBotController.disconnectMyMobileDev;
set(handles.connection_summary, 'String',
handles.mobileBotController.connectionSummary);
guidata(hObject, handles);


% --------------------------------------------------------------------
function data_logging_Callback(hObject, eventdata, handles)
% hObject    handle to data_logging (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function enable_data_logging_Callback(hObject, eventdata, handles)
% hObject    handle to enable_data_logging (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.mobileBotController = handles.mobileBotController.enableDataLogging;
set(handles.logging_status, 'String', handles.mobileBotController.loggingStatus);
guidata(hObject, handles);

% --------------------------------------------------------------------
function disable_data_logging_Callback(hObject, eventdata, handles)
% hObject    handle to disable_data_logging (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.mobileBotController = handles.mobileBotController.disableDataLogging;
set(handles.logging_status, 'String', handles.mobileBotController.loggingStatus);
guidata(hObject, handles);


% --------------------------------------------------------------------
function disable_sensors_Callback(hObject, eventdata, handles)
% hObject    handle to disable_sensors (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.mobileBotController = handles.mobileBotController.disableMobileSensors;
set(handles.accelerometer_status, 'String',
handles.mobileBotController.accelerometerStatus);
set(handles.gyro_sensor_status, 'String',
handles.mobileBotController.gyroSensorStatus);
set(handles.magneto_status, 'String', handles.mobileBotController.magnetoStatus);
set(handles.orientation_status, 'String',
handles.mobileBotController.orientation);
set(handles.position_status, 'String', handles.mobileBotController.position);
guidata(hObject, handles);


% --------------------------------------------------------------------
function enable_sensor_Callback(hObject, eventdata, handles)
% hObject    handle to enable_sensor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function enable_accelerometer_Callback(hObject, eventdata, handles)
% hObject    handle to enable_accelerometer (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.mobileBotController = handles.mobileBotController.enableAccelerometer;
set(handles.accelerometer_status, 'String',
handles.mobileBotController.accelerometerStatus);
guidata(hObject, handles);

% --------------------------------------------------------------------
function enable_gyro_sensor_Callback(hObject, eventdata, handles)
% hObject    handle to enable_gyro_sensor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.mobileBotController = handles.mobileBotController.enableGyroSensor;
set(handles.gyro_sensor_status, 'String',
handles.mobileBotController.gyroSensorStatus);
guidata(hObject, handles);

% --------------------------------------------------------------------
function enable_magnetic_sensor_Callback(hObject, eventdata, handles)
% hObject    handle to enable_magnetic_sensor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.mobileBotController = handles.mobileBotController.enableMagneticSensor;
set(handles.magneto_status, 'String', handles.mobileBotController.magnetoStatus);
guidata(hObject, handles);

% --------------------------------------------------------------------
function enable_orientation_sensor_Callback(hObject, eventdata, handles)
% hObject    handle to enable_orientation_sensor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.mobileBotController = handles.mobileBotController.enableOrientation;
set(handles.orientation_status, 'String',
handles.mobileBotController.orientation);
guidata(hObject, handles);

% --------------------------------------------------------------------
function enable_positon_sensor_Callback(hObject, eventdata, handles)
% hObject    handle to enable_positon_sensor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.mobileBotController = handles.mobileBotController.enablePosition;
set(handles.position_status, 'String', handles.mobileBotController.position);
guidata(hObject, handles);


% --------------------------------------------------------------------
function enable_all_sensors_Callback(hObject, eventdata, handles)
% hObject    handle to enable_all_sensors (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.mobileBotController = handles.mobileBotController.enableAllSensors;
set(handles.accelerometer_status, 'String',
handles.mobileBotController.accelerometerStatus);
set(handles.gyro_sensor_status, 'String',
handles.mobileBotController.gyroSensorStatus);
set(handles.magneto_status, 'String', handles.mobileBotController.magnetoStatus);
set(handles.orientation_status, 'String',
handles.mobileBotController.orientation);
set(handles.position_status, 'String', handles.mobileBotController.position);
guidata(hObject, handles);

% --- Executes on button press in start_data_logging.
function start_data_logging_Callback(hObject, eventdata, handles)
% hObject    handle to start_data_logging (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

try
    while handles.mobileBotController.mobileDevice.Connected == true &&
handles.mobileBotController.mobileDevice.Logging == true...
            && handles.mobileBotController.mobileDevice.AccelerationSensorEnabled
== true...
            &&
handles.mobileBotController.mobileDevice.AngularVelocitySensorEnabled == true...
            && handles.mobileBotController.mobileDevice.MagneticSensorEnabled ==
true...
            && handles.mobileBotController.mobileDevice.OrientationSensorEnabled
== true...
            && handles.mobileBotController.mobileDevice.PositionSensorEnabled ==
true;

        if handles.stop_bit == true;
            handles.stop_bit = false;
            break;
        elseif handles.stop_bit == 0;
            set(handles.sensor_data_log_display, 'String',
matlab.unittest.diagnostics.ConstraintDiagnostic.getDisplayableString(handles.mob
ileBotController.mobileDevice));
            jointAngles = handles.mobileBotController.mobileDevice.Orientation;

            flexVoltage = (readVoltage(handles.mobileBotController.a, 'A0')/5) -
0.1;
            if jointAngles(1,1)/180 >=0.25 && jointAngles(1,1)/180 <= 0.75;
                writePosition(handles.mobileBotController.s1,
jointAngles(1,1)/180); % yaw
            end

            if jointAngles(1,2)/60 >= 0.2 && jointAngles(1,2)/60 <= 0.7;
                writePosition(handles.mobileBotController.s2,
jointAngles(1,2)/60); % pitch
            end

            if jointAngles(1,3)/180 >= 0.25 && jointAngles(1,3)/180 <= 0.75;
```

```
                writePosition(handles.mobileBotController.s3,
jointAngles(1,3)/180); % roll
            end

            if flexVoltage <=0.075
                writePosition(handles.mobileBotController.s4, 0.075); % clutch
            elseif flexVoltage >= 0.2
                writePosition(handles.mobileBotController.s4, 0.2); % declutch
            end

            set(handles.joint_angles, 'String', sprintf('Joint angles : [%f %f
%f]', jointAngles(1,1), jointAngles(1,2), jointAngles(1,3)));
            set(handles.flex_value, 'String', sprintf('Clutch Value : %f',
flexVoltage));

        end

        pause(.075);
        guidata(hObject, handles);
    end
catch
    errordlg('Please enable mobile device connector, connect mobile device and
enable data logging', 'Device not connected !!', 'modal');
end
guidata(hObject, handles);


% --- Executes on button press in stop_data_logging.
function stop_data_logging_Callback(hObject, eventdata, handles)
% hObject    handle to stop_data_logging (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.stop_bit = true;
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function logging_status_CreateFcn(hObject, eventdata, handles)
% hObject    handle to logging_status (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function accelerometer_status_CreateFcn(hObject, eventdata, handles)
% hObject    handle to accelerometer_status (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function gyro_sensor_status_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gyro_sensor_status (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function magneto_status_CreateFcn(hObject, eventdata, handles)
% hObject    handle to magneto_status (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function orientation_status_CreateFcn(hObject, eventdata, handles)
% hObject    handle to orientation_status (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function position_status_CreateFcn(hObject, eventdata, handles)
% hObject    handle to position_status (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function connection_summary_CreateFcn(hObject, eventdata, handles)
% hObject    handle to connection_summary (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on slider movement.
function yaw_Callback(hObject, eventdata, handles)
% hObject    handle to yaw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function yaw_CreateFcn(hObject, eventdata, handles)
% hObject    handle to yaw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
```

```
function pitch_Callback(hObject, eventdata, handles)
% hObject    handle to pitch (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function pitch_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pitch (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function roll_Callback(hObject, eventdata, handles)
% hObject    handle to roll (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function roll_CreateFcn(hObject, eventdata, handles)
% hObject    handle to roll (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes during object creation, after setting all properties.
function sensor_data_log_display_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sensor_data_log_display (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function joint_angles_CreateFcn(hObject, eventdata, handles)
% hObject    handle to joint_angles (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function flex_value_CreateFcn(hObject, eventdata, handles)
% hObject    handle to flex_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --------------------------------------------------------------------
function connect_arduino_Callback(hObject, eventdata, handles)
% hObject    handle to connect_arduino (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function connect_Callback(hObject, eventdata, handles)
% hObject    handle to connect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.mobileBotController = handles.mobileBotController.connectArduino;
guidata(hObject, handles);

% --------------------------------------------------------------------
function disconnect_Callback(hObject, eventdata, handles)
% hObject    handle to disconnect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.mobileBotController = handles.mobileBotController.disConnectArduino;
guidata(hObject, handles);
```

3. Robot and GUI Snapshots

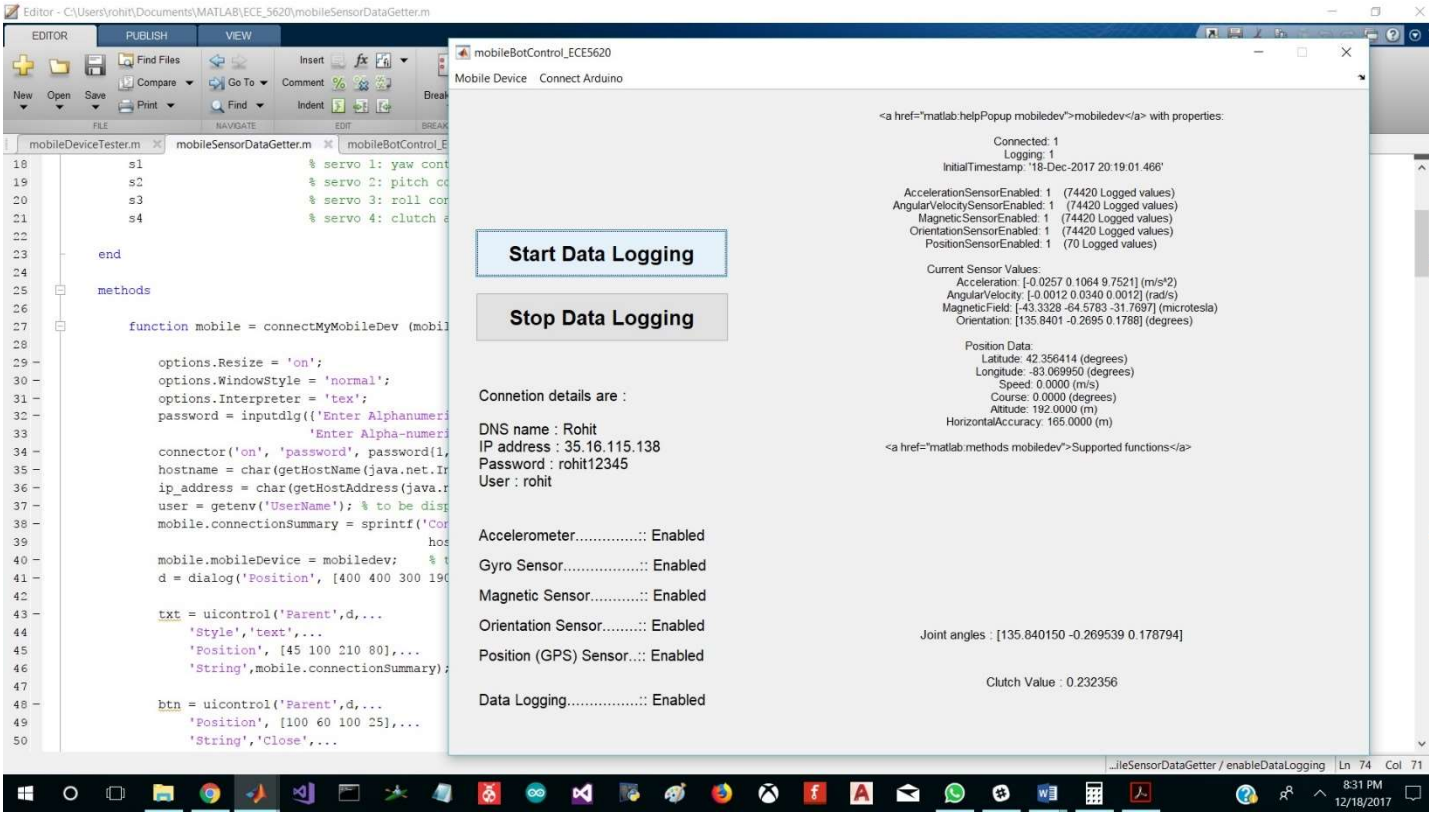


Fig 8: GUI of Remote Controlled Robot

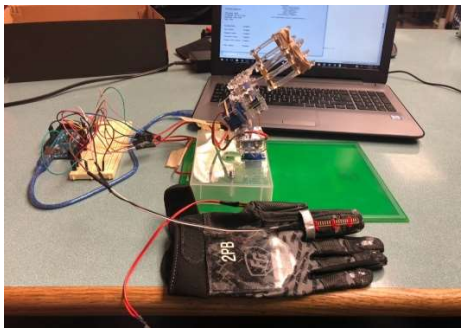


Fig. 9: Complete assembly of robot and Sensors

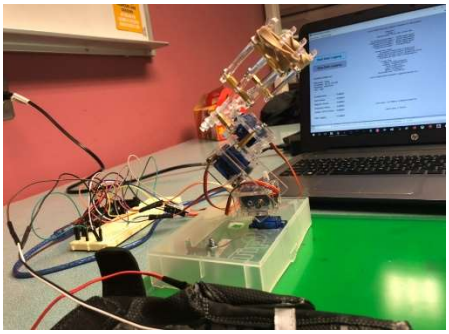


Fig. 10: Side view of Design

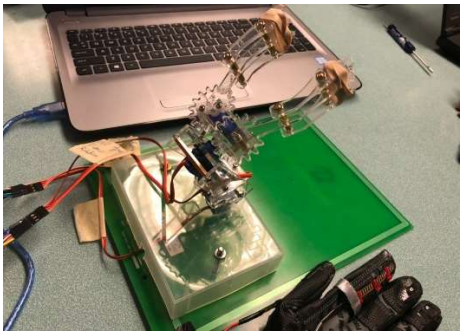


Fig. 11: Top view of Design

G. Data Sheets

1. **Accelerometer: ADXL 345**

   https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf

2. **Flex Sensor**

   https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/FLEX%20SENSOR%20DATA%20SHEET%202014.pdf

3. **HS 53 Micro Servos**

   https://www.servocity.com/hitec-hs-53-servo

H. YouTube Video Link

Please watch the project video demo on YouTube on:

https://youtu.be/FcKlPWJHt0I

I. Program Flow Chart.