Mobile Application Development for IoT Unit-5 Material

Syllabus:

Voice-activated Arduino: Hardware and Software requirements, Writing the Arduino sketch
Bluetooth Low Energy Mobile Robot: Hardware and Software requirements, Writing the Arduino sketch, Enhancing the interface further.

Project-1 Voice-activated Arduino:

Hardware Requirements:

- The Arduino Uno board
- The 5V relay module
- The Adafruit nRF8001 breakout board
- The breadboard
- Jumper wires

Software Requirements:

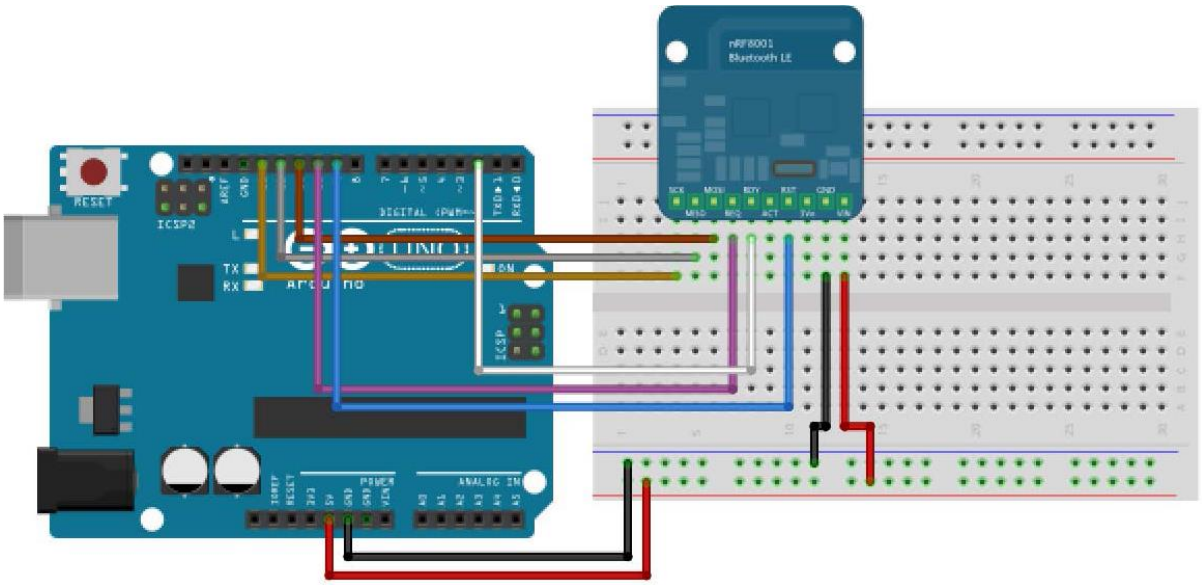
On the software side, you will need the following:
- The Arduino IDE (http://arduino.cc/en/Main/Software)
- The Arduino aREST library (https://github.com/marcoschwartz/aREST/)
- The nRF8001 Arduino library for the BLE chip (https://github.com/adafruit/Adafruit_nRF8001)

To install a given library, simply extract the folder in your Arduino/libraries folder (or create this folder if it doesn't exist yet).

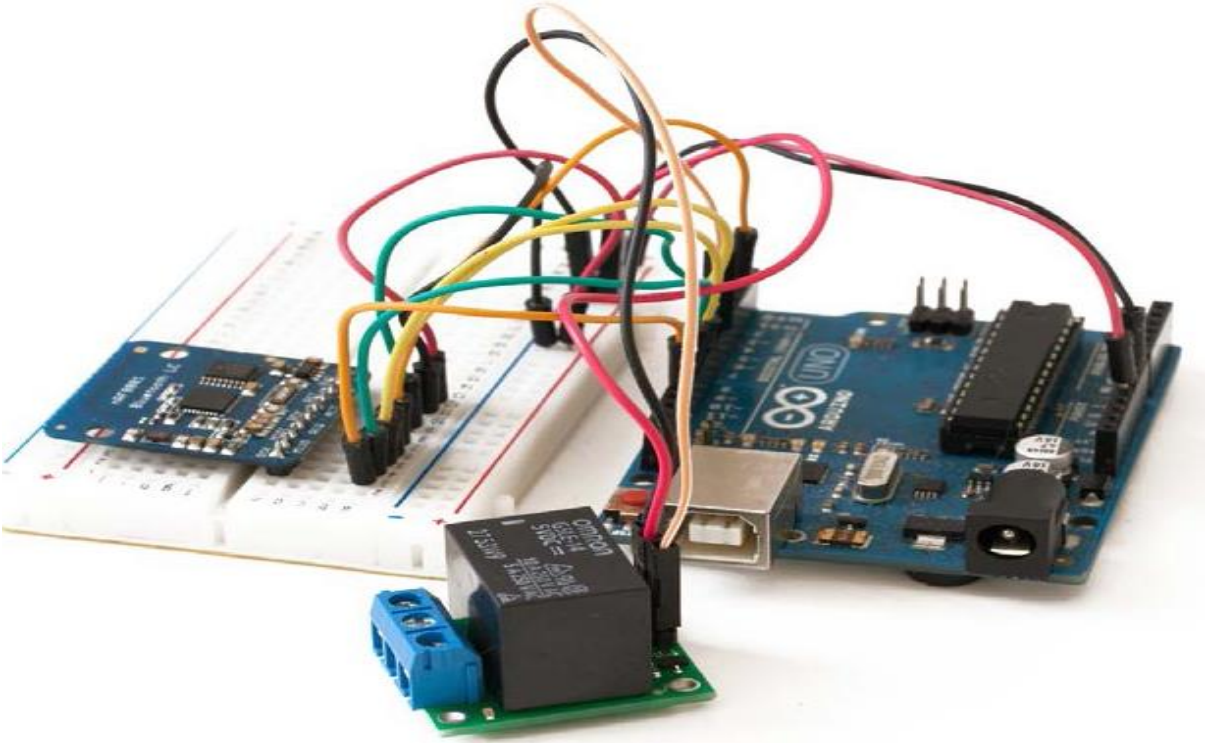
Hardware Configuration

1. Connect the power supply from the Arduino board to the breadboard: 5V of the Arduino board goes to the red power rail, and **GND** goes to the blue power rail.
2. We will now connect the BLE module. First, connect the power supply of the module: **GND** goes to the blue power rail, and **VIN** goes to the red power rail.
3. After this, you need to connect the different wires responsible for the **Serial Peripheral Interface** (**SPI**) communications: **SCK** to Arduino pin **13**, **MISO** to Arduino pin **12**, and **MOSI** to Arduino pin **11**.
4. Then, connect the **REQ** pin to Arduino pin **10**. Finally, connect the **RDY** pin to Arduino pin **2**, and the **RST** pin to Arduino pin **9**.
5. For the relay module, connect the **VCC** pin to the red power rail on the breadboard and the **GND** pin on the blue power rail. Finally, connect the **SIG** pin of the relay to pin number **7** of the Arduino board.

The following is the schematic of the project:



The following is an image of an overview of the assembled project:



Writing the Arduino Sketch:

We will now write the sketch to control the relay from an Android device.

```
// Control Arduino board from BLE
// Libraries
#include <SPI.h>
#include "Adafruit_BLE_UART.h"
#include <aREST.h>
// Pins
#define ADAFRUITBLE_REQ 10
#define ADAFRUITBLE_RDY 2 // Should be pin 2 or 3
```

```
#define ADAFRUITBLE_RST 9
// Relay pin
const int relay_pin = 7;
// Create aREST instance
aREST rest = aREST();
// BLE instance
Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ,
ADAFRUITBLE_RDY, ADAFRUITBLE_RST);
void setup(void)
{
// Start Serial
Serial.begin(115200);
// Start BLE
BTLEserial.begin();
// Give name and ID to device
rest.set_id("001");
rest.set_name("relay_control");
// Init relay pin
pinMode(relay_pin,OUTPUT);
}
void loop() {
// Tell the nRF8001 to do whatever it should be working on.
BTLEserial.pollACI();
// Ask what is our current status
aci_evt_opcode_t status = BTLEserial.getState();
// Handle REST calls
if (status == ACI_EVT_CONNECTED) {
rest.handle(BTLEserial);
}
}
```

Setting up the Android app:

In this project, we will be implementing an Android app that leverages the use of the Speech Recognition API and we are going output that text in an EditText field. In the background, we will also include the BLE services in order to connect to the BLE module and be able to send messages to it. Once we have the BLE and Speech Recognition API set up, we will be able to connect them both by setting up conditions where if the speech is recognized as switch on, it will switch on the relay, whereas if switch off is recognized, the relay will be switched off.

choose the following within the **New Project** setup:
• **Name**: Talk to Arduino
• **Minimum SDK**: 18
• **Project**: Blank Activity
• **Activity Name**: MainScreen
• **Domain**: arduinoandroid.com

Laying out the Android user interface and permissions:

Code for AndroidManifest.xml to give permissions for the app:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.arduinoandroid.androidarduinosensserv" >
<uses-permission android:name="android.hardware.sensor.
gyroscope"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_
ADMIN"/>
<application
android:allowBackup="true"
android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
<activity
android:name=".MainScreen"
android:label="@string/app_name" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.
LAUNCHER" />
</intent-filter>
</activity>
</application>
</manifest>
```

Code for activity_main.xml to create user interface:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
   tools:context=".SpeechActivity">

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Talk to Arduino"
      android:id="@+id/talktoArduino"
      android:layout_centerVertical="true"
      android:layout_centerHorizontal="true" />

   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/recordedTalk"
```

```
        android:text="What is recorded will be written here"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="139dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="Bluetooth Output"
        android:id="@+id/btView"
        android:layout_marginTop="76dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Refresh"
        android:id="@+id/refreshBtn"
        android:layout_above="@+id/talktoArduino"
        android:layout_alignStart="@+id/talktoArduino"
        android:layout_alignEnd="@+id/talktoArduino" />
</RelativeLayout>
```

Code snapshot for MainActivity.java:

```
package com.arduinoandroid.talktoarduino;

import android.app.Activity;
import android.bluetooth.*;
import android.content.Intent;
import android.os.Bundle;
import android.speech.RecognizerIntent;
import android.util.Log;
import android.view.View;
import android.widget.*;
import java.nio.charset.Charset;
import java.util.*;


public class SpeechActivity extends Activity {

    private static final int VOICE_RECOGNITION_REQUEST = 1;

    //Getting the name for Log Tags
    private final String LOG_TAG = SpeechActivity.class.getSimpleName();

    //Declare U.I Elements
    private Button startTalk;
    private Button refresh;
    private EditText speechInput;
    private TextView btv;
```

```
    // UUIDs for UAT service and associated characteristics.
    public static UUID UART_UUID = UUID.fromString("6E400001-B5A3-F393-E0A9-
E50E24DCCA9E");
    public static UUID TX_UUID = UUID.fromString("6E400002-B5A3-F393-E0A9-
E50E24DCCA9E");
    public static UUID RX_UUID = UUID.fromString("6E400003-B5A3-F393-E0A9-
E50E24DCCA9E");
    // UUID for the BTLE client characteristic which is necessary for notifications.
    public static UUID CLIENT_UUID = UUID.fromString("00002902-0000-1000-8000-
00805f9b34fb");

    // BTLE stateta
    private BluetoothAdapter adapter;
    private BluetoothGatt gatt;
    private BluetoothGattCharacteristic tx;
    private BluetoothGattCharacteristic rx;

    private boolean areServicesAccessible = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_speech);
        startTalk = (Button) findViewById(R.id.talktoArduino);
        refresh = (Button) findViewById(R.id.refreshBtn);
        speechInput = (EditText) findViewById(R.id.recordedTalk);
        btv = (TextView) findViewById(R.id.btView);

        startTalk.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                recordSpeech();
            }
        });

        refresh.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                restartScan();
            }
        });

    }

    public void recordSpeech() {

        Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);

        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
```

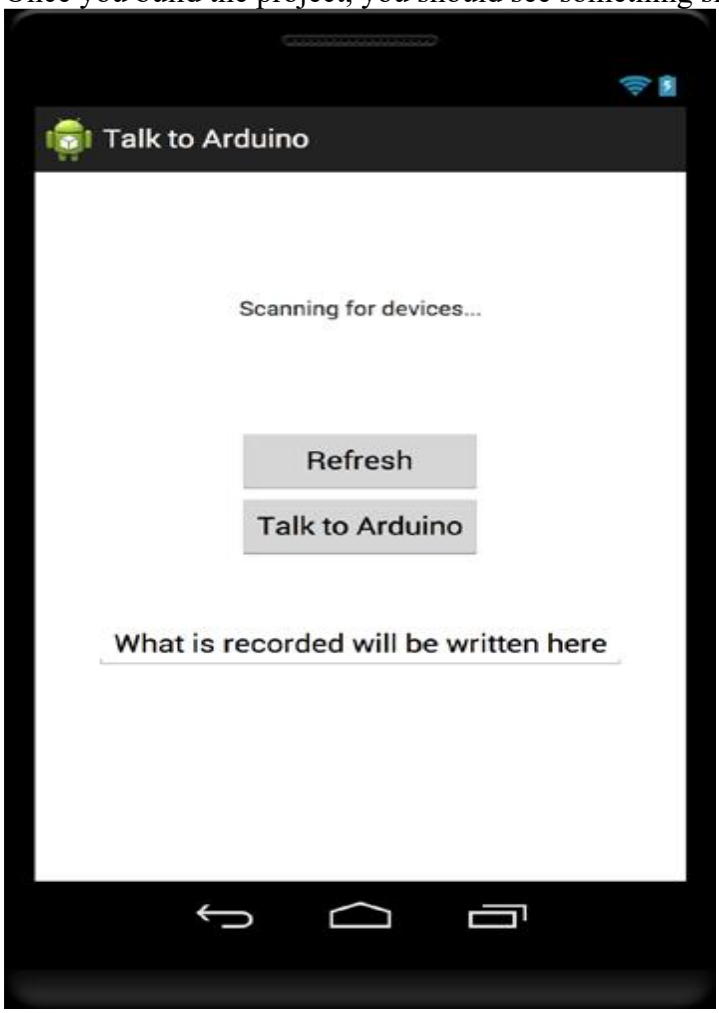
```
        intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "You can now send a command to
the Arduino");

        startActivityForResult(intent, VOICE_RECOGNITION_REQUEST);
    }


    @Override

    protected void onActivityResult(int requestCode, int resultCode, Intent data) {

        if (requestCode == VOICE_RECOGNITION_REQUEST && resultCode ==
RESULT_OK) {

            ArrayList<String> matches =
data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);

            String userInput = matches.get(0);

            TextView textSaid = (TextView) findViewById(R.id.recordedTalk);

            textSaid.setText(matches.get(0));

            //add an if else loop or case statement

            if (userInput.equalsIgnoreCase("switch on")) {
                String setOutputMessage = "/digital/7/1 /";
                tx.setValue(setOutputMessage.getBytes(Charset.forName("UTF-8")));
                if (gatt.writeCharacteristic(tx)) {
                    writeSensorData("Sent: " + setOutputMessage);
                } else {
                    writeSensorData("Couldn't write TX characteristic!");
                }
            } else if (userInput.equalsIgnoreCase("switch off")) {
                String setOutputMessage = "/digital/7/0 /";
                tx.setValue(setOutputMessage.getBytes(Charset.forName("UTF-8")));
                if (gatt.writeCharacteristic(tx)) {
                    writeSensorData("Sent: " + setOutputMessage);
                } else {
                    writeSensorData("Couldn't write TX characteristic!");
                }
            }

        }
        super.onActivityResult(requestCode, resultCode, data);
    }

    private void writeSensorData(final CharSequence text) {
        Log.e(LOG_TAG, text.toString());
        btv.setText(text.toString());
    }

    // BTLE device scanning bluetoothGattCallback.
```

```
    // Main BTLE device bluetoothGattCallback where much of the logic occurs.
    private BluetoothGattCallback bluetoothGattCallback = new BluetoothGattCallback() {
        // Called whenever the device connection state changes, i.e. from disconnected to
connected.
        @Override
        public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
            super.onConnectionStateChange(gatt, status, newState);
            if (newState == BluetoothGatt.STATE_CONNECTED) {
                writeSensorData("Connected!");
                // Discover services.
                if (!gatt.discoverServices()) {
                    writeSensorData("Failed to start discovering services!");
                }
            } else if (newState == BluetoothGatt.STATE_DISCONNECTED) {
                writeSensorData("Disconnected!");
            } else {
                writeSensorData("Connection state changed.  New state: " + newState);
            }
        }

        // Called when services have been discovered on the remote device.
        // It seems to be necessary to wait for this discovery to occur before
        // manipulating any services or characteristics.
        public void onServicesDiscovered(BluetoothGatt gatt, int status) {
            super.onServicesDiscovered(gatt, status);
            if (status == BluetoothGatt.GATT_SUCCESS) {
                writeSensorData("Service discovery completed!");
            } else {
                writeSensorData("Service discovery failed with status: " + status);
            }
            // Save reference to each characteristic.
            tx = gatt.getService(UART_UUID).getCharacteristic(TX_UUID);
            rx = gatt.getService(UART_UUID).getCharacteristic(RX_UUID);

            // Setup notifications on RX characteristic changes (i.e. data received).
            // First call setCharacteristicNotification to enable notification.
            if (!gatt.setCharacteristicNotification(rx, true)) {
                writeSensorData("Couldn't set notifications for RX characteristic!");
            }

            // Next update the RX characteristic's client descriptor to enable notifications.
            if (rx.getDescriptor(CLIENT_UUID) != null) {
                BluetoothGattDescriptor desc = rx.getDescriptor(CLIENT_UUID);
                desc.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
                if (!gatt.writeDescriptor(desc)) {
                    writeSensorData("Couldn't write RX client descriptor value!");
                }
            } else {
                writeSensorData("Couldn't get RX client descriptor!");
            }
            areServicesAccessible = true;
        }
```

```
};


protected void onStart() {
    Log.d(LOG_TAG,"onStart has been called");
    super.onStart();
    // / Scan for all BTLE devices.
    // The first one with the UART service will be chosen--see the code in the scanCallback.
    adapter = BluetoothAdapter.getDefaultAdapter();
    startScan();
}

//When this Activity isn't visible anymore
protected void onStop() {
    Log.d(LOG_TAG,"onStop has been called");
    //disconnect and close Bluetooth Connection for better reliability
    if (gatt != null) {
        gatt.disconnect();
        gatt.close();
        gatt = null;
        tx = null;
        rx = null;
    }
    super.onStop();
}

//BLUETOOTH METHODS
private void startScan() {
    if (!adapter.isEnabled()) {
        adapter.enable();
    }
    if (!adapter.isDiscovering()) {
        adapter.startDiscovery();
    }
    writeSensorData("Scanning for devices...");
    adapter.startLeScan(scanCallback);
}

private void stopScan() {
    if (adapter.isDiscovering()) {
        adapter.cancelDiscovery();
    }
    writeSensorData("Stopping scan");
    adapter.stopLeScan(scanCallback);
}

private void restartScan() {
    stopScan();
    startScan();
}

/**
 * Main callback following an LE device scan
```

```
     */
    private BluetoothAdapter.LeScanCallback scanCallback = new
BluetoothAdapter.LeScanCallback() {
        // Called when a device is found.
        @Override
        public void onLeScan(BluetoothDevice bluetoothDevice, int i, byte[] bytes) {
            Log.d(LOG_TAG, bluetoothDevice.getAddress());

            writeSensorData("Found device: " + bluetoothDevice.getAddress());

            // Check if the device has the UART service.
            if (BluetoothUtils.parseUUIDs(bytes).contains(UART_UUID)) {
                // Found a device, stop the scan.
                adapter.stopLeScan(scanCallback);
                writeSensorData("Found UART service!");
                // Connect to the device.
                // Control flow will now go to the bluetoothGattCallback functions when BTLE
events occur.
                gatt = bluetoothDevice.connectGatt(getApplicationContext(), false,
bluetoothGattCallback);
            }
        }
    };
}
```

Once you build the project, you should see something similar to the following screenshot:



Project-2: Bluetooth Low Energy Mobile Robot:

Introduction:

The base of this project is of course the robot itself. For this project, we used a DFRobot miniQ two-wheeled robot chassis. It comes with a round robot chassis, two DC motors, two wheels, and some screws and bolts so that you can mount multiple Arduino boards on it. You can basically use any equivalent robot chassis that has two wheels coupled with DC motors and on which you can mount Arduino-compatible boards.

To control the robot, we are actually going to use three different Arduino boards. The "brain" of the robot will be a simple Arduino Uno board. On top of that, we will use a DFRobot motor shield to control the two DC motors of the robot. And on top of these two boards, we will put a prototyping shield so that we can connect different modules to the robot.

To control the robot remotely, we will again use BLE. To give BLE connectivity to the robot, we used an Adafruit nRF8001 breakout board. To give the robot the ability to detect what is in front of it, we added an URM37 ultrasonic sensor to the project. As we will see, this sensor is really easy to interface with Arduino.

Finally, you will also need some jumper wires to make the different connections between the robot, the sensor, and the Bluetooth module.

Hardware Requirements:

• An Arduino Uno board
• An Arduino motor shield
• An Arduino prototyping shield
• An nRF8001 breakout board
• An ultrasonic range sensor
• An ultrasonic sensor mounting kit
• A DFRobot miniQ chassis
• A 7.4 V battery
• Jumper wires

Software Requirements:

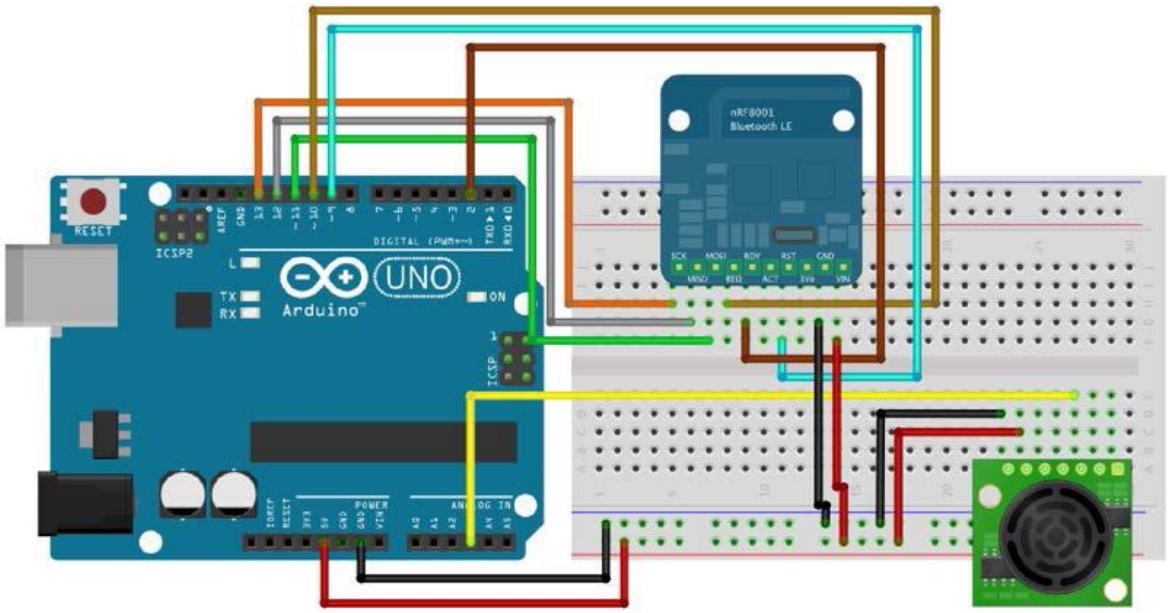
• A library for the nRF8001 chip (https://github.com/adafruit/Adafruit_nRF8001)
• The aREST library to send commands to the robot (https://github.com/marcoschwartz/aREST)

Hardware Configuration:

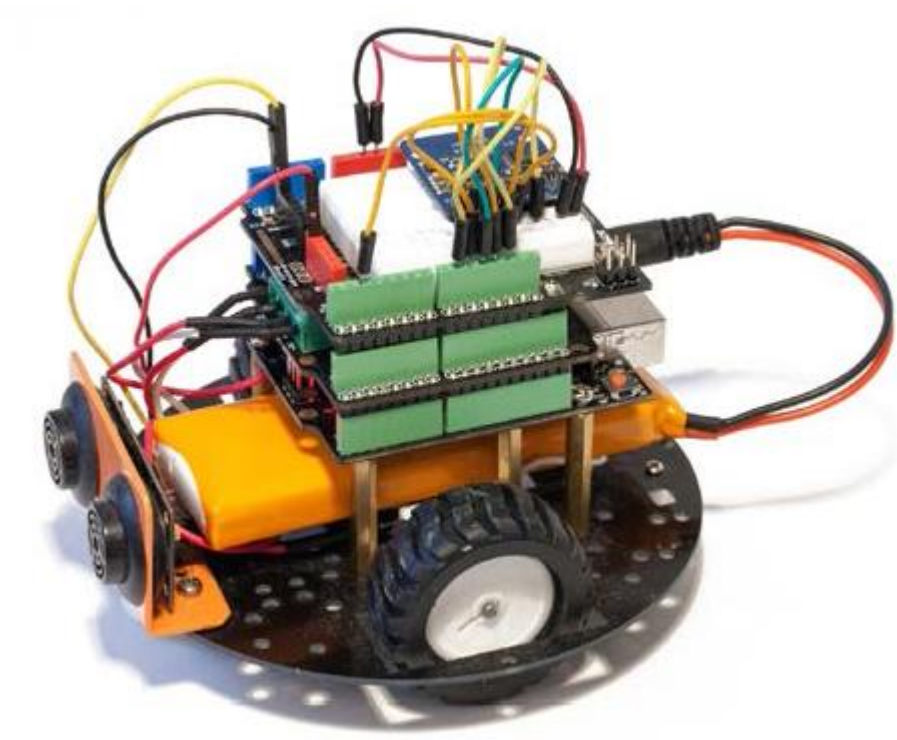
1. First, we are now going to connect the BLE module.
2. Place the module on the prototyping shield.
3. Connect the power supply of the module as follows: **GND** goes to the prototyping shield's **GND** pin, and **VIN** goes to the prototyping shield's +5V.
4. After that, you need to connect the different wires responsible for the SPI interface: **SCK** to Arduino pin **13**, **MISO** to Arduino pin **12**, and **MOSI** to Arduino pin **11**.
5. Then connect the **REQ** pin to Arduino pin **10**.

6. Finally, connect the **RDY** pin to Arduino pin **2** and the **RST** pin to Arduino pin **9**.
7. For the URM37 module, connect the **VCC** pin of the module to Arduino +5V, **GND** to **GND**, and the **PWM** pin to the Arduino **A3** pin.
8. Finally, connect the 7.4 V battery to the Arduino Uno board power jack.
The battery is simply placed below the Arduino Uno board.

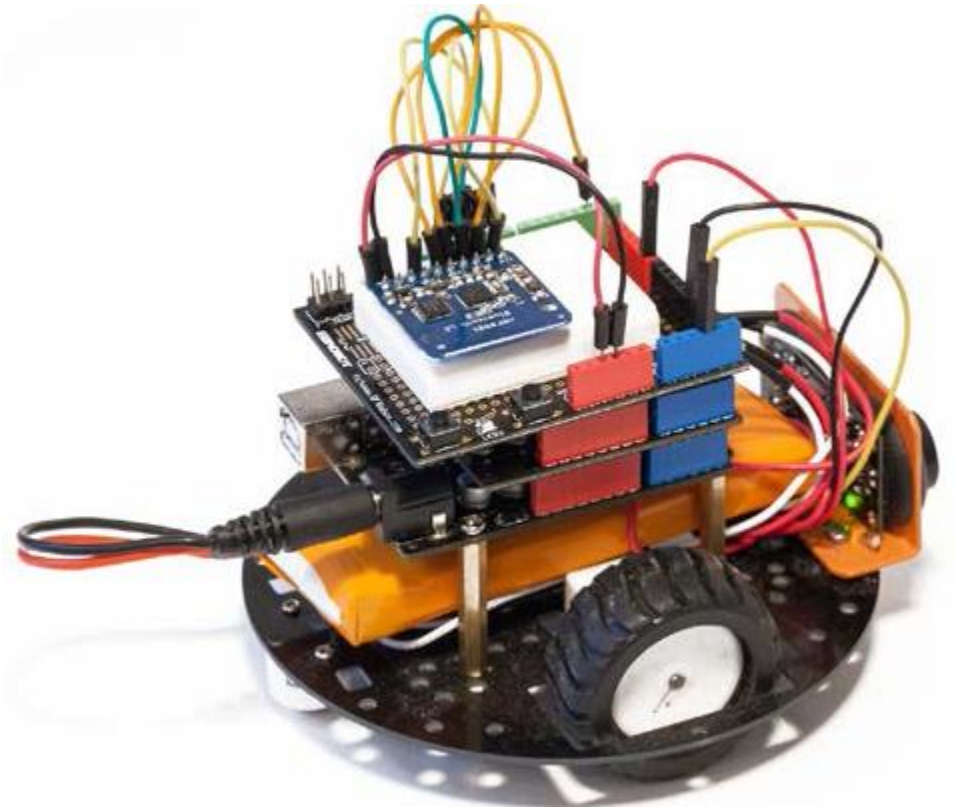
The following image shows the schematic of the project:



The following is a front-view image of the robot when fully assembled:

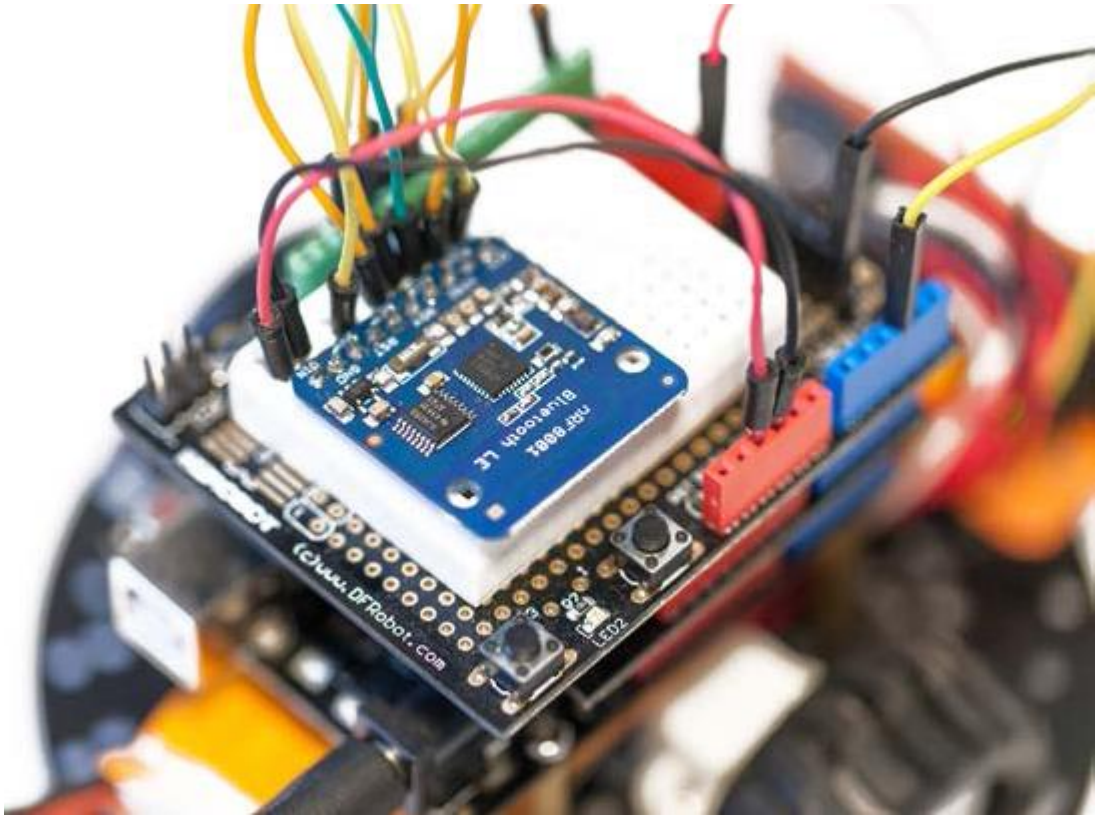


The following image shows the back of the robot when fully assembled



The first step is to assemble the robot chassis. Then, you need to attach the different Arduino boards and shields to the robot. Use the spacers found in the robot chassis kit to mount the Arduino Uno board first. Then put the Arduino motor shield on top of that. At this point, use the screw header terminals to connect the two DC motors to the motor shield. This is how it should look at this point:

Finally, mount the prototyping shield on top of the motor shield.
We are now going to connect the BLE module and the ultrasonic sensor to the Arduino prototyping shield. The following is a close-up image of the prototyping shield with the BLE module connected:



Testing the robot:

We are now going to write a sketch to test the different functionalities of the robot, first without using Bluetooth. As the sketch is quite long, we will look at the code piece by piece. Before you proceed, make sure that the battery is always plugged into the robot. Now perform the following steps:

Arduino Sketch for Testing the robot:

```
// Robot test via aREST + Serial

// Libraries
#include <aREST.h>

// Motor pins
int speed_motor1 = 6;
int speed_motor2 = 5;
int direction_motor1 = 7;
int direction_motor2 = 4;

// Sensor pins
int distance_sensor = A3;

// Create aREST instance
aREST rest = aREST();
```

```
// Variable to be exposed to the API
int distance;

void setup(void)
{
  // Start Serial
  Serial.begin(115200);

  // Expose variables to REST API
  rest.variable("distance",&distance);

  // Expose functions
  rest.function("forward",forward);
  rest.function("backward",backward);
  rest.function("left",left);
  rest.function("right",right);
  rest.function("stop",stop);

  // Give name and ID to device
  rest.set_id("001");
  rest.set_name("mobile_robot");
}

void loop() {

  // Measure distance
  distance = measure_distance(distance_sensor);

  // Handle REST calls
  rest.handle(Serial);

}

// Forward
int forward(String command) {

  send_motor_command(speed_motor1,direction_motor1,100,1);
  send_motor_command(speed_motor2,direction_motor2,100,1);
  return 1;
}

// Backward
int backward(String command) {

  send_motor_command(speed_motor1,direction_motor1,100,0);
  send_motor_command(speed_motor2,direction_motor2,100,0);
  return 1;
}

// Left
int left(String command) {
```

```
  send_motor_command(speed_motor1,direction_motor1,75,0);
  send_motor_command(speed_motor2,direction_motor2,75,1);
  return 1;
}

// Right
int right(String command) {

  send_motor_command(speed_motor1,direction_motor1,75,1);
  send_motor_command(speed_motor2,direction_motor2,75,0);
  return 1;
}

// Stop
int stop(String command) {

  send_motor_command(speed_motor1,direction_motor1,0,1);
  send_motor_command(speed_motor2,direction_motor2,0,1);
  return 1;
}

// Function to command a given motor of the robot
void send_motor_command(int speed_pin, int direction_pin, int pwm, boolean dir)
{
  analogWrite(speed_pin,pwm); // Set PWM control, 0 for stop, and 255 for maximum speed
  digitalWrite(direction_pin,dir);
}

// Measure distance from the ultrasonic sensor
int measure_distance(int pin){

  unsigned int Distance=0;
  unsigned long DistanceMeasured=pulseIn(pin,LOW);

  if(DistanceMeasured==50000){              // the reading is invalid.
     Serial.print("Invalid");
   }
    else{
     Distance=DistanceMeasured/50;      // every 50us low level stands for 1cm
   }

  return Distance;
}
```

We are now going to test the robot. Before you do anything, ensure that the battery is always plugged into the robot. This will ensure that the motors are not trying to get power from your computer USB port, which could damage it.

Also place some small support at the bottom of the robot so that the wheels don't touch the ground. This will ensure that you can test all the commands of the robot without the robot moving too far from your computer, as it is still attached via the USB cable. Now you can upload the sketch to your Arduino Uno board. Open the serial monitor and type the following:

/forward

This should make both the wheels of the robot turn in the same direction. You can also try the other commands to move the robot to make sure they all work properly. Then, test the ultrasonic distance sensor by typing the following:
/distance

You should get back the distance (in centimeters) in front of the sensor:
{"distance": 24, "id": "001", "name": "mobile_robot", "connected": true}
Try changing the distance by putting your hand in front of the sensor and typing the command again.

Writing the complete Arduino Sketch:

```
// Robot test via aREST + Serial

// Libraries
#include <SPI.h>
#include "Adafruit_BLE_UART.h"
#include <aREST.h>

// BLE pins
#define ADAFRUITBLE_REQ 10
#define ADAFRUITBLE_RDY 2     // This should be an interrupt pin, on Uno thats #2 or #3
#define ADAFRUITBLE_RST 9

// Motor pins
int speed_motor1 = 6;
int speed_motor2 = 5;
int direction_motor1 = 7;
int direction_motor2 = 4;

// Sensor pins
int distance_sensor = A3;

// BLE instance
Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ,
ADAFRUITBLE_RDY, ADAFRUITBLE_RST);

// Create aREST instance
aREST rest = aREST();

// Variable to be exposed to the API
int distance;

void setup(void)
{
  // Start Serial
  Serial.begin(115200);
  BTLEserial.begin();

  // Expose variables to REST API
  rest.variable("distance",&distance);

  // Expose functions
  rest.function("forward",forward);
```

```
  rest.function("backward",backward);
  rest.function("left",left);
  rest.function("right",right);
  rest.function("stop",stop);

  // Give name and ID to device
  rest.set_id("001");
  rest.set_name("mobile_robot");
}

aci_evt_opcode_t laststatus = ACI_EVT_DISCONNECTED;

void loop() {

  // Measure distance
  distance = measure_distance(distance_sensor);

  // Tell the nRF8001 to do whatever it should be working on.
  BTLEserial.pollACI();

  // Ask what is our current status
  aci_evt_opcode_t status = BTLEserial.getState();
  // If the status changed....
  if (status != laststatus) {
    // print it out!
    if (status == ACI_EVT_DEVICE_STARTED) {
        Serial.println(F("* Advertising started"));
    }
    if (status == ACI_EVT_CONNECTED) {
        Serial.println(F("* Connected!"));
    }
    if (status == ACI_EVT_DISCONNECTED) {
        Serial.println(F("* Disconnected or advertising timed out"));
    }
    // OK set the last status change to this one
    laststatus = status;
  }

  // Handle REST calls
  if (status == ACI_EVT_CONNECTED) {
    rest.handle(BTLEserial);
  }

}

// Forward
int forward(String command) {

  send_motor_command(speed_motor1,direction_motor1,200,1);
  send_motor_command(speed_motor2,direction_motor2,200,1);
  return 1;
}
```

```
// Backward
int backward(String command) {

  send_motor_command(speed_motor1,direction_motor1,200,0);
  send_motor_command(speed_motor2,direction_motor2,200,0);
  return 1;
}

// Left
int left(String command) {

  send_motor_command(speed_motor1,direction_motor1,150,0);
  send_motor_command(speed_motor2,direction_motor2,150,1);
  return 1;
}

// Right
int right(String command) {

  send_motor_command(speed_motor1,direction_motor1,150,1);
  send_motor_command(speed_motor2,direction_motor2,150,0);
  return 1;
}

// Stop
int stop(String command) {

  send_motor_command(speed_motor1,direction_motor1,0,1);
  send_motor_command(speed_motor2,direction_motor2,0,1);
  return 1;
}

// Function to command a given motor of the robot
void send_motor_command(int speed_pin, int direction_pin, int pwm, boolean dir)
{
  analogWrite(speed_pin,pwm); // Set PWM control, 0 for stop, and 255 for maximum speed
  digitalWrite(direction_pin,dir);
}

// Measure distance from the ultrasonic sensor
int measure_distance(int pin){

  unsigned int Distance=0;
  unsigned long DistanceMeasured=pulseIn(pin,LOW);

  if(DistanceMeasured==50000){              // the reading is invalid.
     Serial.print("Invalid");
   }
    else{
     Distance=DistanceMeasured/50;      // every 50us low level stands for 1cm
   }

  return Distance;    }
```

Setting up the Android app:

choose the following within the **New Project** setup
• **Name**: Mobile Robot
• **Minimum SDK**: 18
• **Project**: Blank Activity
• **Activity Name**: RobotControlActivity
• **Domain**: arduinoandroid.com

Laying out the Android user interface and setting permissions:

Code for AndroidManifest.xml to give permissions for the app:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.arduinoandroid.androidarduinosensserv" >
<uses-permission android:name="android.hardware.sensor.
gyroscope"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_
ADMIN"/>
<application
android:allowBackup="true"
android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
<activity
android:name=".MainScreen"
android:label="@string/app_name" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.
LAUNCHER" />
</intent-filter>
</activity>
</application>
</manifest>
```

Code for activity_main.xml to create user interface:

```
<LinearLayout
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   xmlns:android="http://schemas.android.com/apk/res/android">

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Connect"
      android:id="@+id/connectBtn"
      android:layout_gravity="center_horizontal"
```

```
    android:background="@drawable/buttonshape"
    android:layout_margin="10dp"/>

<Button
    style="?android:attr/buttonStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Forward"
    android:id="@+id/fwdBtn"
    android:layout_gravity="center_horizontal"
    android:background="@drawable/buttonshape"
    android:layout_margin="10dp"/>

<LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="57dp">

    <Button
        style="?android:attr/buttonStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Left"
        android:id="@+id/leftBtn"
        android:layout_weight="1"
        android:background="@drawable/buttonshape"
        android:layout_marginRight="10dp"
        android:layout_marginLeft="10dp"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Stop"
        android:id="@+id/stopBtn"
        android:layout_gravity="center_horizontal"
        android:layout_weight="1"
        android:background="@drawable/buttonshape"
        />

    <Button
        style="?android:attr/buttonStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Right"
        android:id="@+id/rightBtn"
        android:layout_weight="1"
        android:background="@drawable/buttonshape"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"/>

</LinearLayout>
```

```
<Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Backward"
        android:id="@+id/backwardBtn"
        android:layout_gravity="center_horizontal"
        android:background="@drawable/buttonshape"
        android:layout_margin="10dp"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Connection Status View"
        android:id="@+id/connectionStsView"
        android:layout_gravity="center_horizontal"
        />

</LinearLayout>
```

Code snapshot for MainActivity.java:

```
package com.arduinoandroid.mobilerobot;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattDescriptor;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import com.arduinoandroid.mobilerobot.Bluetooth.BluetoothUtils;

import java.nio.charset.Charset;
import java.util.UUID;


public class RobotControlActivity extends Activity {

    //User Interface Elements
    Button fwdBtn;
    Button leftBtn;
    Button rightBtn;
    Button backBtn;
    Button stopBtn;
    Button connectBtn;
```

```
    TextView connectionSts;

    //Logging Variables
    private final String LOG_TAG = RobotControlActivity.class.getSimpleName();

    // UUIDs for UAT service and associated characteristics.
    public static UUID UART_UUID = UUID.fromString("6E400001-B5A3-F393-E0A9-
E50E24DCCA9E");
    public static UUID TX_UUID = UUID.fromString("6E400002-B5A3-F393-E0A9-
E50E24DCCA9E");
    public static UUID RX_UUID = UUID.fromString("6E400003-B5A3-F393-E0A9-
E50E24DCCA9E");

    // UUID for the BTLE client characteristic which is necessary for notifications.
    public static UUID CLIENT_UUID = UUID.fromString("00002902-0000-1000-8000-
00805f9b34fb");

    // BTLE stateta
    private BluetoothAdapter adapter;
    private BluetoothGatt gatt;
    private BluetoothGattCharacteristic tx;
    private BluetoothGattCharacteristic rx;

    private boolean areServicesAccessible = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_robot_control);

        fwdBtn = (Button) findViewById(R.id.fwdBtn);
        leftBtn = (Button) findViewById(R.id.leftBtn);
        rightBtn = (Button) findViewById(R.id.rightBtn);
        backBtn = (Button) findViewById(R.id.backwardBtn);
        stopBtn = (Button) findViewById(R.id.stopBtn);
        connectBtn = (Button) findViewById(R.id.connectBtn);

        connectionSts = (TextView)findViewById(R.id.connectionStsView);

        fwdBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String setOutputMessage = "/forward /";
                tx.setValue(setOutputMessage.getBytes(Charset.forName("UTF-8")));
                if (gatt.writeCharacteristic(tx)) {
                    writeConnectionData("Sent: " + setOutputMessage);
                } else {
                    writeConnectionData("Couldn't write TX characteristic!");
                }
            }
        });

        leftBtn.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
    public void onClick(View view) {
        String setOutputMessage = "/left /";
        tx.setValue(setOutputMessage.getBytes(Charset.forName("UTF-8")));
        if (gatt.writeCharacteristic(tx)) {
            writeConnectionData("Sent: " + setOutputMessage);
        } else {
            writeConnectionData("Couldn't write TX characteristic!");
        }
    }
});

rightBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String setOutputMessage = "/right /";
        tx.setValue(setOutputMessage.getBytes(Charset.forName("UTF-8")));
        if (gatt.writeCharacteristic(tx)) {
            writeConnectionData("Sent: " + setOutputMessage);
        } else {
            writeConnectionData("Couldn't write TX characteristic!");
        }
    }
});
backBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String setOutputMessage = "/backward /";
        tx.setValue(setOutputMessage.getBytes(Charset.forName("UTF-8")));
        if (gatt.writeCharacteristic(tx)) {
            writeConnectionData("Sent: " + setOutputMessage);
        } else {
            writeConnectionData("Couldn't write TX characteristic!");
        }
    }
});
stopBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String setOutputMessage = "/stop /";
        tx.setValue(setOutputMessage.getBytes(Charset.forName("UTF-8")));
        if (gatt.writeCharacteristic(tx)) {
            writeConnectionData("Sent: " + setOutputMessage);
        } else {
            writeConnectionData("Couldn't write TX characteristic!");
        }
    }
});

connectBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
```

```
            restartScan();
        }
    });
}


@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.robot_control, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
    if (id == R.id.action_settings) {
        return true;
    }
    return super.onOptionsItemSelected(item);
}

private void writeConnectionData(final CharSequence text) {
    Log.e(LOG_TAG, text.toString());
    connectionSts.setText(text.toString());
}

// BTLE device scanning bluetoothGattCallback.

// Main BTLE device bluetoothGattCallback where much of the logic occurs.
private BluetoothGattCallback bluetoothGattCallback = new BluetoothGattCallback() {
    // Called whenever the device connection state changes, i.e. from disconnected to
connected.
    @Override
    public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
        super.onConnectionStateChange(gatt, status, newState);
        if (newState == BluetoothGatt.STATE_CONNECTED) {
            writeConnectionData("Connected!");
            // Discover services.
            if (!gatt.discoverServices()) {
                writeConnectionData("Failed to start discovering services!");
            }
        } else if (newState == BluetoothGatt.STATE_DISCONNECTED) {
            writeConnectionData("Disconnected!");
        } else {
            writeConnectionData("Connection state changed.  New state: " + newState);
        }
    }

    // Called when services have been discovered on the remote device.
```

```
    // It seems to be necessary to wait for this discovery to occur before
    // manipulating any services or characteristics.
    public void onServicesDiscovered(BluetoothGatt gatt, int status) {
        super.onServicesDiscovered(gatt, status);
        if (status == BluetoothGatt.GATT_SUCCESS) {
            writeConnectionData("Service discovery completed!");
        } else {
            writeConnectionData("Service discovery failed with status: " + status);
        }
        // Save reference to each characteristic.
        tx = gatt.getService(UART_UUID).getCharacteristic(TX_UUID);
        rx = gatt.getService(UART_UUID).getCharacteristic(RX_UUID);

        // Setup notifications on RX characteristic changes (i.e. data received).
        // First call setCharacteristicNotification to enable notification.
        if (!gatt.setCharacteristicNotification(rx, true)) {
            writeConnectionData("Couldn't set notifications for RX characteristic!");
        }

        // Next update the RX characteristic's client descriptor to enable notifications.
        if (rx.getDescriptor(CLIENT_UUID) != null) {
            BluetoothGattDescriptor desc = rx.getDescriptor(CLIENT_UUID);
            desc.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
            if (!gatt.writeDescriptor(desc)) {
                writeConnectionData("Couldn't write RX client descriptor value!");
            }
        } else {
            writeConnectionData("Couldn't get RX client descriptor!");
        }
        areServicesAccessible = true;
    }
};


protected void onStart() {
    Log.d(LOG_TAG,"onStart has been called");
    super.onStart();
    // / Scan for all BTLE devices.
    // The first one with the UART service will be chosen--see the code in the scanCallback.
    adapter = BluetoothAdapter.getDefaultAdapter();
    startScan();
}

//When this Activity isn't visible anymore
protected void onStop() {
    Log.d(LOG_TAG,"onStop has been called");
    //disconnect and close Bluetooth Connection for better reliability
    if (gatt != null) {
        gatt.disconnect();
        gatt.close();
        gatt = null;
        tx = null;
        rx = null;
```

```
        }
        super.onStop();
    }

    //BLUETOOTH METHODS
    private void startScan() {
        if (!adapter.isEnabled()) {
            adapter.enable();
        }
        if (!adapter.isDiscovering()) {
            adapter.startDiscovery();
        }
        writeConnectionData("Scanning for devices...");
        adapter.startLeScan(scanCallback);
    }

    private void stopScan() {
        if (adapter.isDiscovering()) {
            adapter.cancelDiscovery();
        }
        writeConnectionData("Stopping scan");
        adapter.stopLeScan(scanCallback);
    }

    private void restartScan() {
        stopScan();
        startScan();
    }

    /**
     * Main callback following an LE device scan
     */
    private BluetoothAdapter.LeScanCallback scanCallback = new
BluetoothAdapter.LeScanCallback() {
        // Called when a device is found.
        @Override
        public void onLeScan(BluetoothDevice bluetoothDevice, int i, byte[] bytes) {
            Log.d(LOG_TAG, bluetoothDevice.getAddress());

            writeConnectionData("Found device: " + bluetoothDevice.getAddress());

            // Check if the device has the UART service.
            if (BluetoothUtils.parseUUIDs(bytes).contains(UART_UUID)) {
                // Found a device, stop the scan.
                adapter.stopLeScan(scanCallback);
                writeConnectionData("Found UART service!");
                // Connect to the device.
                // Control flow will now go to the bluetoothGattCallback functions when BTLE
events occur.
                gatt = bluetoothDevice.connectGatt(getApplicationContext(), false,
bluetoothGattCallback);
            }
        }
```

```
    };
}
```

Once you build the project, you should see something similar to the following screenshot: