ECE 435: Introduction to Smart Devices

Connecting and Sending Gyroscope Data to a Selected Bluetooth Device

By: Michael Bowyer

Professor: Paul Watta

March 15, 2017
Winter 2017

Honor Code:

I have neither given nor received unauthorized assistance on this graded report.

X____Michael Bowyer_____

Introduction

The tutorial presented in this report is meant to help guide users through two specific tasks related to creating Android applications. These two functions are communicating with a remote Bluetooth device and retrieving gyroscope data. This report also touches on sending the gyroscope data over Bluetooth communication. The combination of these two functions can be used to control a remote Bluetooth device using the internal gyroscope of the android device.

Background information

Some prerequisite knowledge must be defined and understood before attempting to work through this tutorial. The following terms needed to be understood to fully understand the following tutorial.

Unpaired Bluetooth Devices: Devices which are capable of Bluetooth communication, but have never been paired with the android device being used. These devices must be discovered before being paired

Paired Bluetooth Devices: Devices which are capable of Bluetooth communication, and have been discovered by the android device being used. Pairing happens after the android device and the remote Bluetooth device have exchanged necessary information to establish a communication channel.

Connected Bluetooth Devices: Devices which are capable of Bluetooth communication, and have previously been paired with the android device. These devices and the android device have an “active” connection in which the two devices can communicate to one another.

MAC Address: Known as Media access control address (MAC address) of a device is a unique identifier assigned to network interfaces for communications at the data link layer of a network segment. Each device Bluetooth device has a unique MAC Address which is used to identify itself.

Permissions

The first step in creating an application which is capable of connecting and communicating with a remote Bluetooth device, it is necessary for the app to have those permissions. For any application to utilize the android devices Bluetooth radio these permissions must be declared in the AndroidManifest.XML. The following are the two required permissions the application must have in order to communicate with the android devices onboard Bluetooth radio.

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

Establishing Bluetooth Connection

Before diving into connecting to the Bluetooth communication portion of this tutorial is should be noted that the Bluetooth interfaces which Android offers are slightly disorganized. These is not one consolidated Bluetooth library which encapsulates the necessary functions to communicate via Bluetooth. A Bluetooth class was created to created do just that, encapsulate the setup and

communication handling of Bluetooth communication. This class can be found in the code found the in-reference section. [1]

This class has many functions which allow users to setup a Bluetooth connection and then send data the connection. But the class is rather limited, and has been edited for functionality. The additions to the class can be found in the GitHub repository made for this tutorial [2].

Now the first step to connecting to Bluetooth is to setup a few variables within the MainActivity Class to be used in various function calls. These variables are below.

```
//BLUETOOTH RELATED VARIABLES
Bluetooth myBT;
ArrayList<String> deviceList;
ArrayAdapter<String> deviceListAdapter;
public final int REQUEST_ENABLE_BT = 1;
```

The myBT object is a reference to the Bluetooth class which handles the connection to and communication with the Bluetooth device. All Bluetooth related functions can be called by using the instance of this object which is initialized in the OnCreate of the MainActivity. The Array list and Adapter are used to store the list of devices which the device the app is running on has been paired with.

Please note that this Class does not have functionality to set the device as “discoverable”, nor does it have the ability to scan for devices that are discoverable in the surrounding area. It is assumed that the user of the application has already “paired” the devices using the devices built in Bluetooth settings menu. This allows the user to connect to the device without the possibility of the app crashing, as the built-in Android Bluetooth setting is more stable, and efficient in connection.

Assuming the user has already paired their device previous to starting the app, The user must first create an instance of the Bluetooth Class object. This object requires an activity and a Handler to be created. The handler which was created specifically for this class in the main activity can be seen below. The main purpose of the handler is to send information from the threads which handle Bluetooth communication back to the main activity.

```
//initalize Bluetooth handler and get device list
myBT = new Bluetooth(this,mHandler);// create instance of bluetooth object
deviceList=myBT.getDeviceList();//save list of paired devices.
```

The Handler itself is created in the MainActivity, and is assigned a function called handle message which is called every time the Bluetooth status has changed. The Bluetooth status changes when the device is connecting, becomes connected, a message is sent (or fails to send), or a message is received from the remote device. The purpose of this is to just send this information from the multiple threads which are running via the Bluetooth object to the main activity so that the main activity can display the information mentioned above.

```
//Handle messages which are sent from bluetooth threads
private final Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case Bluetooth.MESSAGE_STATE_CHANGE://when state is changed IE
connected, COnnected, disconnected.
                Log.d("myDebug", "MESSAGE_STATE_CHANGE: " + msg.arg1);
                break;
            case Bluetooth.MESSAGE_WRITE:// This device attempted to send a message
to the remote device
                Log.d("myDebug", "MESSAGE_WRITE ");
                break;
            case Bluetooth.MESSAGE_READ://this dewice recieved a message, and saved
it in the myBTRecievedString
                Log.d("myDebug", "MESSAGE_READ =" + myBT.recievedString);
                updateRecievedDataTextView(myBT.recievedString);
                break;
            case Bluetooth.MESSAGE_DEVICE_NAME://the device name when the device
has become connected
                Log.d("myDebug", "MESSAGE_DEVICE_NAME "+myBT.connectedDeviceName);
                break;
            case Bluetooth.MESSAGE_TOAST://when a message failed to send.
                Log.d("myDebug", "MESSAGE_TOAST "+msg);
                break;
        }
    }
};
```

Now that the necessary Bluetooth object has been created, the device list has been obtained, and a message handler has been created it is now possible to connect to a remote Bluetooth device which is selected by the user. The list of connected devices is still held in the **deviceList** list, which just holds a list of the device names, and not any of the device information (that information is encapsulated in the Bluetooth object). To connect to one of these devices a function must be called, and the devices name (in string format) must be called. Because it is difficult to specify the device name exactly from a user entering in the device name using a keyboard, it was opted to create a ListView to display all of the devices, and using an OnClickListener to determine which device from the list of devices the user selects to connect to the device.

```
deviceListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long
id) {
        String selectedDeviceName = deviceList.get(position);//get device name
        connectService(selectedDeviceName);//start connection with device
    }
});
```

Once the device name is selected from the ListView, the devicename string is retrieved, and then passed in to the connectService function, which is defined in the MainActivity.Java. This function is responsible for the main activities activation of the Bluetooth capability if the device currently has it disabled. The code to do so is shown below.

```
public void connectService(String deviceName){//Creates connection to remote device
    try {
        StatusView.setText("Connecting to "+deviceName);//update activity of
connection status
        BluetoothAdapter bluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();//grab bluetooth adapter of the device(this is
BT radio of the device)
        if (bluetoothAdapter.isEnabled()) {//check if BT is enabled on the device.
            myBT.start();//start bluetooth connection in BT object
            myBT.connectDevice(deviceName);//connect to device
            Log.v("myDebug", "Btservice started - listening");
            StatusView.setText("Connected to "+ myBT.connectedDeviceName);//update
activity of connection status
        } else {//bleutooth is not enabled.
            Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE); //create an intent to request to
turn on bluetooth
            startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);// start
bluetooth if application enableBTIntent returns true
            Log.w("myDebug", "Btservice started - bluetooth is not enabled,
requesting access");
            StatusView.setText("Bluetooth Not enabled");//update activity of
connection status
        }
    } catch(Exception e){
        Log.v("myDebug", "Unable to start bt ",e);
        StatusView.setText("Unable to connect " +e);
    }
}
```

TO check if Bluetooth is enabled, the activity must have access to the devices Bluetooth radio, which can be accessed through the BluetoothAdapter. The Bluetooth adapter class has a built in function which check to see if the Bluetooth functionality is enabled. This is function is the isEnabled function. If the device does not have the Bluetooth enabled, a new intent is created which prompts the user to enable Bluetooth on the device.

If the device already has Bluetooth enabled then the start and connectDevice functions are called. The Start function starts the communication service. Specifically starts a session in listening (server) mode for which it listens for a connection. The connectDevice retrieves the devices MAC address and establishes the connection between the android device and the remote device. This entire process is handled on its own thread, and will throw an exception if the connection fails.

Once the connection has been established to the remote device the message handler will receive a message stating so. From then on out, it is up to the MainActivity to handle what happens when the user wants to send data, and how to display received data when it is received. Below is an

example of what could be done to send and receive data.

```
//////////*BLUETOOTH FUNCTIONS*/////////////
public void SendBTData(View v)//send bluetooth data
{
    String str_send = dataToWrite.getText().toString();//create string to send from
EditText
    myBT.sendMessage(str_send);//send the string
}
public void updateRecievedDataTextView(String rec_str)//update textview of recieved
data
{
    RecievedDataView.setText(rec_str);//update textview
}
```

As described above in the message handler, the message handler function receives a message from the remote device. When a message is received by the android device from the remote Bluetooth device the function updateRecievedDataTextView(**myBT.recievedString**); is called. This function can be seen above, which simply updated a textview titled "RecievedDataView.

When the user would like to send data on some event the function myBT.sendMessage(string) is called. This function simply sends a string to the remote Bluetooth device. In example shown above, the application retrieves the data which entered into an EditText field, and sends it and time the SendBTData function is called. This function (not shown) is associated with a button in the main activity, and the OnClick activity which occurs when clicked is the SendBTData. The sendMessage function will allow the user to send information, such as gyroscope data, to the remote device.

Getting Sensor Data from a Gyroscope

Now that the device is capable of sending Data over Bluetooth it is time to get data from the sensor. There is one "master" class which handles sensor interfaces, and will allow users to retrieve all of the information about the devices sensors. That class is the Sensor Manager, and an instance of this class is saved in the main activity.

```
//SENSOR RELATED VARIABLES
private SensorManager sManager;
```

The first step in the onCreate method in the activity is to create an object of the sensor manager by calling the getSystemService function. This function will store a reference to the devices sensor manager so the application can access it's sensor data.

```
//Initialize Sensor Manager
sManager = (SensorManager) getSystemService(SENSOR_SERVICE);
```

Once the sensorManager has been established, it is desired to create an event that occurs each time the desired sensor (gyroscope) is changed. It is also desired to broadcast this data as fast as possible, so that when controlling a remote device, it can get a continuous stream of the sensors data, not just when it is changed. But this is only desired when the application is being used! Not when the android device is using another app! Because of this the following code snippets were used.

```
////////////*SENSOR FUNCTIONS*////////////
//when this Activity starts
@Override
protected void onResume()
{
    super.onResume();
    /*register the sensor listener to listen to the gyroscope sensor, use the
    callbacks defined in this class, and gather the sensor information as quick
    as possible*/
    sManager.registerListener(this,
sManager.getDefaultSensor(Sensor.TYPE_ORIENTATION),SensorManager.SENSOR_DELAY_FASTE
ST);
}

//When this Activity isn't visible anymore
@Override
protected void onStop()
{
    //unregister the sensor listener
    sManager.unregisterListener(this);
    super.onStop();
}

@Override
public void onSensorChanged(SensorEvent event)
{
    //if sensor is unreliable, return void
    if (event.accuracy == SensorManager.SENSOR_STATUS_UNRELIABLE)
    {
        return;
    }

    //else it will output the Roll, Pitch and Yawn values
    GyroData.setText("Orientation X (Roll) :"+ Float.toString(event.values[2])
+"\n"+"Orientation Y (Pitch) :"+ Float.toString(event.values[1]) +"\n"+
"Orientation Z (Yaw) :"+ Float.toString(event.values[0]));
}
```

The onResume function is the function which occurs each time the user of the android app brings the application to the forefront. This can happen when the app is opened initially, or when the user switches back to this application. The onStop function does just the opposite, it is called anytime the user puts the application in the background.

These two functions are necessary to start and pause the listeners which listen to the sensors. To explain further, the desired behavior is to monitor the sensors when the user is using the application, and not to monitor the sensors when the user is not using the application. It can be seen that in these functions, the “listeners are registered”. In the OnResume function, a listener is created with listens to a sensor of type Orientation (Gyroscope), and is called at SENSOR_DELAY_FASTEST which attempts to read the sensor as fast as possible. In the OnStop function all listeners occurring on this activity are “unregistered” meaning that they are no longer going to listen to the sensors which were previously registered. Note that the sensor can be change from Orientation to other sensor functions such as gravity, acceleration, humidity, etc. and the sample rate can also be changed. More about this information can be found in the android documentation [3].

Now that the gyroscope event is being listened to onResume, an event must be set up to occur whenever the sensor data is read. This function is the onSensorChanged function which monitors SensorEvents. This function simple checks if the accuracy of the sensor is reliable, and if it is reliable, then the data which is read from the sensor is simply printed to a TextView called GyroData.

Sending Sensor Data Over Bluetooth

Now that the device can send and receive Bluetooth data as well as retrieve information about the gyroscope sensor, the two functions can be easily combined. Using the sendMessage function described in the Bluetooth section can be called, and any string which contains the sensor data can be sent via Bluetooth. In this instance, the sensor data which was printed to a TextView is now being transmitted over Bluetooth!

```
myBT.sendMessage("Orientation X (Roll) :"+ Float.toString(event.values[2]) +"\n"+
        "Orientation Y (Pitch) :"+ Float.toString(event.values[1]) +"\n"+
        "Orientation Z (Yaw) :"+ Float.toString(event.values[0]));//send the string
```

Exercises

1. Create a List view of all the Bluetooth devices which the device is paired with.
2. Print Gyroscope data to a text view.
3. Print Received Bluetooth data to a text view.
4. Send Bluetooth data to remote device whenever a button is pressed.

Bibliography

[1] Coconauts, "Arduino-Android-Bluetooth Github," 2015. [Online]. Available: https://github.com/coconauts/Arduino-Android-Bluetooth. [Accessed 15 March 2017].

[2] M. Bowyer, "mikebowyer/ECE450-Tutorial," 16 March 2017. [Online]. Available: https://github.com/mikebowyer/ECE450-Tutorial.

[3] A. Developers, "Sensor Overview," [Online]. Available: https://developer.android.com/guide/topics/sensors/sensors_overview.html. [Accessed 15 March 2017].