Project Report

Project Name: Firebird V operation using Android app

Interns Names:

1.Archie Mittal
mittal.archie93@gmail.com
9453010442
2.Kanupriya Sharma
kanupriyasharma94@gmail.com
9899895782

Project Mentor:

1.Vibhu Mishra
vibhumishra94@gmail.com
9970665887
2.Rutuja Ekatpure
rattzzz07@gmail.com
9665924979
3.Mehul Makwana
mehul.makwan@live.com
9405624900

Duration of the internship:

25/05/2015 to 08/07/2015

Abstract: This project is all about bringing all the features of Firebird V on our finger tips through our Android phone.We are able to perform all the basic operations of our robot namely motion control,getting the readings of all the sensors,changing the velocity and gesture control via our smart phone.We have achieved this communication with robot wirelessly with the help of bluetooth module.

Objective of the work: The objective of the project was to develop an Android app which can be used for motion control, velocity control and sensor monitoring of Firebird V.The first task was to understand the basic concepts of Java and Android then make a basic Hello World app.The next task was to interface the bluetooth module with the robot.Then we had to make an Android app to perform basic I/O operation on Firebird V for e.g. turning the buzzer ON and OFF.Then we had to add motion control to our app.Furthermore we had to get the readings of the sensors from the robot to our Android device.Then we had to control the motor's velocity with the help of our phone.Later we had to work on our UI and integrate all these features with the main app.Lastly we had to test our app on different versions of Android OS.

Completion: We have successfully completed our project with all the tasks done along with some add on feature i.e controlling the robot with hand gestures.

Results and Discussion: Now we finally have an android app for our robot.This app will make controlling of robot very easy.It will also develop an interest among kids as well due to its interactive UI. We have made our Android project on Android Studio.

- Android is a mobile operating system (OS) based on the Linux kernel and currently developed by Google. With a user interface based on direct manipulation, Android is designed primarily for touchscreen mobile devices such as smartphones and tablet computers, with specialized user interfaces for televisions (Android TV), cars (Android Auto), and wrist watches (Android Wear).
- The OS uses touch inputs that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard. Despite being primarily designed for touchscreen input, it has also been used in game consoles, digital cameras, regular PCs, and other electronics. As of 2015, Android has the largest installed base of all operating systems.

Android Studio

- Android Studio is the official IDE for Android application development, based on IntelliJ IDEA.
- At the core of Android Studio is an intelligent code editor capable of advanced code completion, refactoring, and code analysis.
- The powerful code editor helps you be a more productive Android app developer. On top of the capabilities you expect from IntelliJ, Android Studio offers:

Intelligent code editor

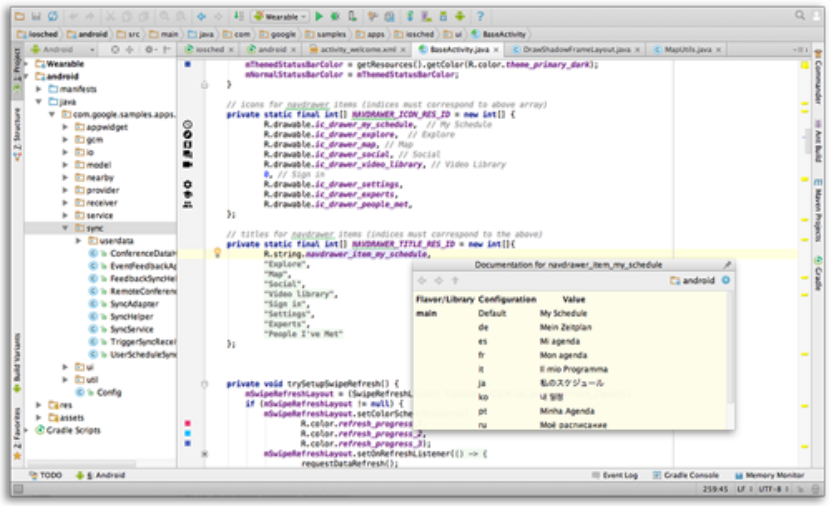

Figure 1:

- At the core of Android Studio is an intelligent code editor capable of advanced code completion, refactoring, and code analysis.

  The powerful code editor helps you be a more productive Android app developer.

  Code templates and GitHub integration

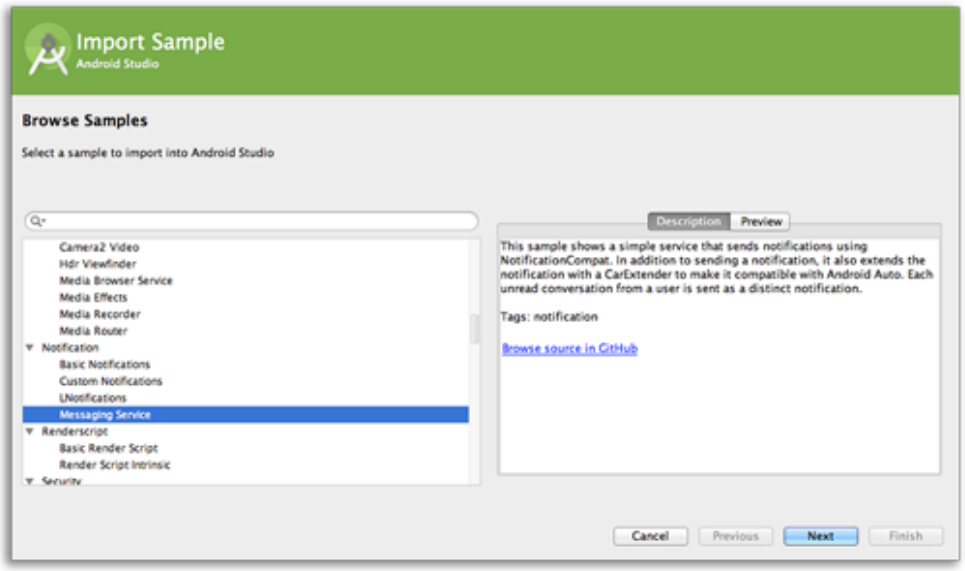

Figure 2:

- New project wizards make it easier than ever to start a new project.

  Start projects using template code for patterns such as navigation drawer and view pagers, and even import Google code samples from GitHub.

  Multi-screen app development

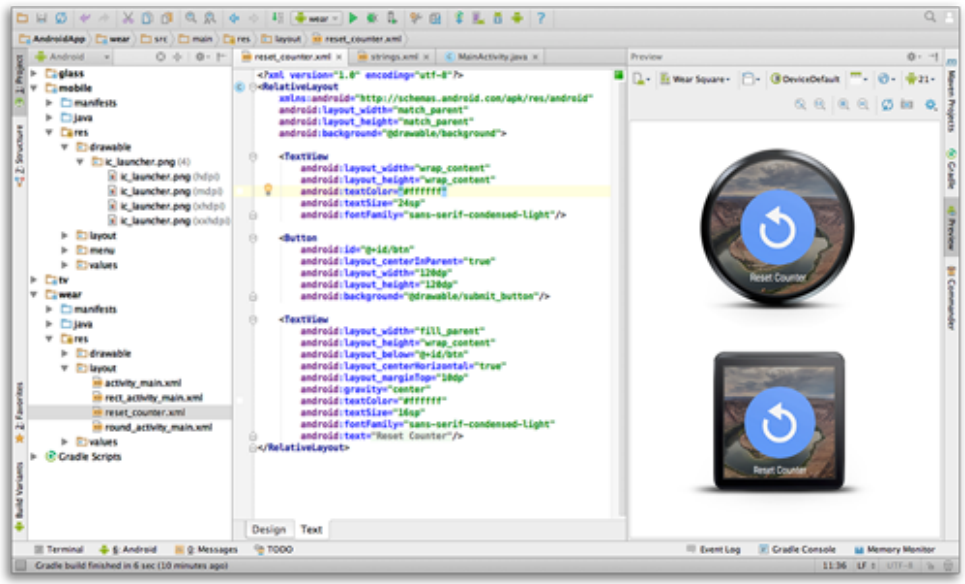

Figure 3:

- Build apps for Android phones, tablets, Android Wear, Android TV, Android Auto and Google Glass.

  With the new Android Project View and module support in Android Studio, it's easier to manage app projects and resources.

Virtual devices for all shapes and sizes

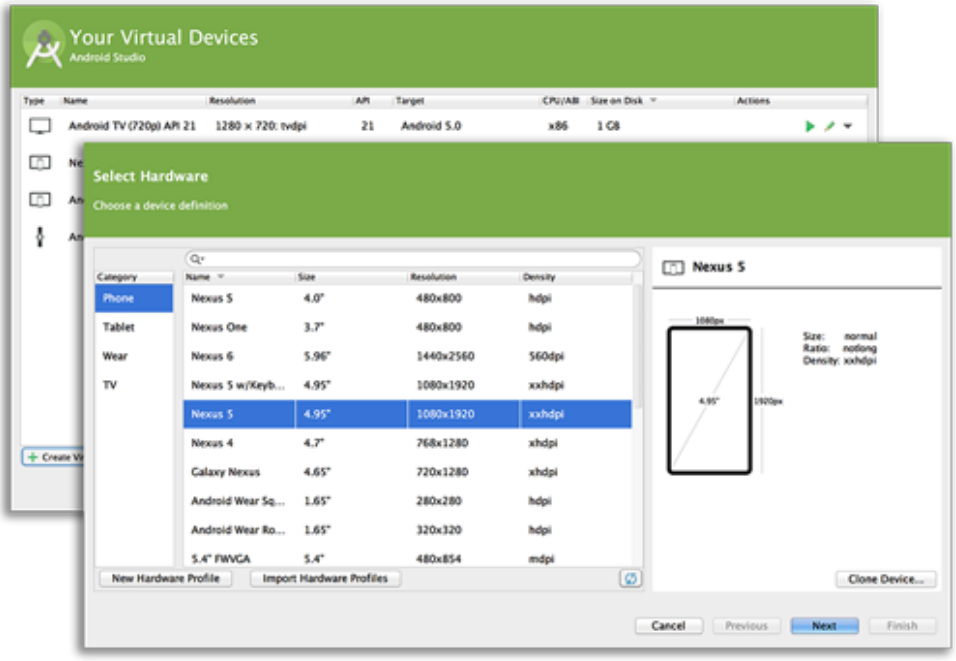

Figure 4:

- Android Studio comes pre-configured with an optimized emulator image.
  The updated and streamlined Virtual Device Manager provides pre-defined device profiles for common Android devices.

Android builds evolved, with Gradle

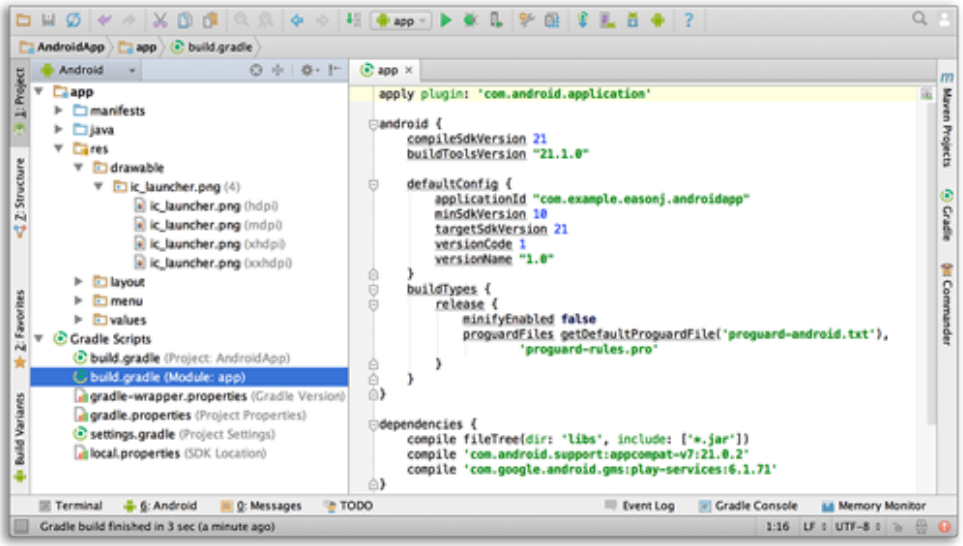

Figure 5:

- Create multiple APKs for your Android app with different features using the same project.
  Manage app dependencies with Maven.
  Build APKs from Android Studio or the command line.

More about Android Studio

- Built on IntelliJ IDEA Community Edition, the popular Java IDE by JetBrains.
- Flexible Gradle-based build system
- Build variants and multiple apk file generation
- Code templates to help you build common app features
- Rich layout editor with support for drag and drop theme editing
- lint tools to catch performance, usability, version compatibility, and other problems
- ProGuard and app-signing capabilities
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine And much more

System Requirements

Windows

- Microsoft® Windows® 8/7/Vista/2003 (32 or 64-bit)
- 2 GB RAM minimum, 4 GB RAM recommended
- 400 MB hard disk space
- At least 1 GB for Android SDK, emulator system images, and caches 1280 x 800 minimum screen resolution
- Java Development Kit (JDK) 7
- Optional for accelerated emulator: Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality

Mac OS X

- Mac® OS X® 10.8.5 or higher, up to 10.9 (Mavericks)
- 2 GB RAM minimum, 4 GB RAM recommended
- 400 MB hard disk space
- At least 1 GB for Android SDK, emulator system images, and caches
- 1280 x 800 minimum screen resolution
- Java Runtime Environment (JRE) 6
- Java Development Kit (JDK) 7
- Optional for accelerated emulator: Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality
- On Mac OS, run Android Studio with Java Runtime Environment (JRE) 6 for optimized font rendering. You can then configure your project to use Java Development Kit (JDK) 6 or JDK 7.

Linux

- GNOME or KDE desktop
- GNU C Library (glibc) 2.15 or later
- 2 GB RAM minimum, 4 GB RAM recommended
- 400 MB hard disk space
- At least 1 GB for Android SDK, emulator system images, and caches
- 1280 x 800 minimum screen resolution
- Oracle® Java Development Kit (JDK) 7

Bluetooth Connectivity

- The Android platform includes support for the Bluetooth network stack, which allows a device to wirelessly exchange data with other Bluetooth devices. The application framework provides access to the Bluetooth functionality through the Android Bluetooth APIs. These APIs let applications wirelessly connect to other Bluetooth devices, enabling point-to-point and multipoint wireless features.

  Using the Bluetooth APIs, an Android application can perform the following:

- Scan for other Bluetooth devices
- Query the local Bluetooth adapter for paired Bluetooth devices
- Establish RFCOMM channels
- Connect to other devices through service discovery
- Transfer data to and from other devices
- Manage multiple connections

- All of the Bluetooth APIs are available in the android.bluetooth package. Here's a summary of the classes and interfaces you will need to create Bluetooth connections:

  BluetoothAdapter

  Represents the local Bluetooth adapter (Bluetooth radio). The BluetoothAdapter is the entry-point for all Bluetooth interaction. Using this, you can discover other Bluetooth devices, query a list of bonded (paired) devices, instantiate a BluetoothDevice using a known MAC address, and create a BluetoothServerSocket to listen for communications from other devices.

  BluetoothDevice

  Represents a remote Bluetooth device. Use this to request a connection with a remote device through a BluetoothSocket or query information about the device such as its name, address, class, and bonding state.

  BluetoothSocket

  Represents the interface for a Bluetooth socket (similar to a TCP Socket). This is the connection point that allows an application to exchange data with another Bluetooth device via InputStream and OutputStream.

  BluetoothServerSocket

  Represents an open server socket that listens for incoming requests (similar to a TCP ServerSocket). In order to connect two Android devices, one device must open a server socket with this class. When a remote Bluetooth device makes a connection request to the this device, the BluetoothServerSocket will return a connected BluetoothSocket when the connection is accepted.

  BluetoothClass

  Describes the general characteristics and capabilities of a Bluetooth device. This is a read-only set of properties that define the device's major and minor device classes and its services. However, this does not reliably describe all Bluetooth profiles and services supported by the device, but is useful as a hint to the device type.

  BluetoothProfile

  An interface that represents a Bluetooth profile. A Bluetooth profile is a wireless interface specification for Bluetooth-based communication between devices. An example is the Hands-Free profile.

  BluetoothHeadset

  Provides support for Bluetooth headsets to be used with mobile phones. This includes both Bluetooth Headset and Hands-Free (v1.5) profiles.

  BluetoothA2dp

  Defines how high quality audio can be streamed from one device to another over a Bluetooth connection. "A2DP" stands for Advanced Audio Distribution Profile.

BluetoothHealth

Represents a Health Device Profile proxy that controls the Bluetooth service.

BluetoothHealthCallback

An abstract class that you use to implement BluetoothHealth callbacks. You must extend this class and implement the callback methods to receive updates about changes in the application?s registration state and Bluetooth channel state.

BluetoothHealthAppConfiguration

Represents an application configuration that the Bluetooth Health third-party application registers to communicate with a remote Bluetooth health device.

BluetoothProfile.ServiceListener

An interface that notifies BluetoothProfile IPC clients when they have been connected to or disconnected from the service (that is, the internal service that runs a particular profile).

Bluetooth Permissions

- In order to use Bluetooth features in your application, you must declare the Bluetooth permission BLUETOOTH. You need this permission to perform any Bluetooth communication, such as requesting a connection, accepting a connection, and transferring data.
- If you want your app to initiate device discovery or manipulate Bluetooth settings, you must also declare the BLUETOOTH _ADMIN permission. Most applications need this permission solely for the ability to discover local Bluetooth devices. The other abilities granted by this permission should not be used, unless the application is a "power manager" that will modify Bluetooth settings upon user request. Note: If you use BLUETOOTH _ADMIN permission, then you must also have the BLUETOOTH permission.

  Declare the Bluetooth permission(s) in your application manifest file. For example:

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

Figure 6:

Setting Up Bluetooth

Before your application can communicate over Bluetooth, you need to verify that Bluetooth is supported on the device, and if so, ensure that it is enabled.

If Bluetooth is not supported, then you should gracefully disable any Bluetooth features. If Bluetooth is supported, but disabled, then you can request that the user enable Bluetooth without leaving your application. This setup is accomplished in two steps, using the BluetoothAdapter.

1, Get the BluetoothAdapter

The BluetoothAdapter is required for any and all Bluetooth activity. To get the BluetoothAdapter, call the static getDefaultAdapter() method. This returns a BluetoothAdapter that represents the device's own Bluetooth adapter (the Bluetooth radio). There's one Bluetooth adapter for the entire system, and your application can interact with it using this object. If getDefaultAdapter() returns null, then the device does not support Bluetooth and your story ends here. For example:

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
    // Device does not support Bluetooth
}
```

Figure 7:

2.Enable Bluetooth

Next, you need to ensure that Bluetooth is enabled. Call isEnabled() to check whether Bluetooth is currently enable. If this method returns false, then Bluetooth is disabled. To request that Bluetooth be enabled, call startActivityForResult() with the ACTION_REQUEST_ENABLE action Intent. This will issue a request to enable Bluetooth through the system settings (without stopping your application). For example: A dialog will appear requesting user permission to enable Bluetooth. If

```
if (!mBluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

Figure 8:

the user responds "Yes," the system will begin to enable Bluetooth and focus will return to your application once the process completes (or fails).
The REQUEST_ENABLE_BT constant passed to startActivityForResult() is a locally defined integer (which must be greater than 0), that the system passes back to you in your onActivityResult() implementation as the requestCode parameter.

If enabling Bluetooth succeeds, your activity receives the RESULT_OK result code in the onActivityResult() callback. If Bluetooth was not enabled due to an error (or the user responded "No") then the result code is RESULT_CANCELED.

Optionally, your application can also listen for the ACTION_STATE _CHANGED broadcast Intent, which the system will broadcast whenever the Bluetooth state has changed. This broadcast contains the extra fields EXTRA_STATE and EXTRA _PREVIOUS _STATE, containing the new and old Bluetooth states, respectively. Possible values for these extra fields are STATE _TURNING_ON, STATE_ON, STATE_TURNING_OFF, and STATE_OFF. Listening for this broadcast can be useful to detect changes made to the Bluetooth state while your app is running.

Finding Devices

Using the BluetoothAdapter, you can find remote Bluetooth devices either through device discovery or by querying the list of paired (bonded) devices.

Device discovery is a scanning procedure that searches the local area for Bluetooth enabled devices and then requesting some information about each one (this is sometimes referred to as "discovering," "inquiring" or "scanning"). However, a Bluetooth device within the local area will respond to a discovery request only if it is currently enabled to be discoverable. If a device is discoverable, it will respond to the discovery request by sharing some information, such as the device name, class, and its unique MAC address. Using this information, the device performing discovery can then choose to initiate a connection to the discovered device.

Once a connection is made with a remote device for the first time, a pairing request is automatically presented to the user. When a device is paired, the basic information about that device (such as the device name, class, and MAC address) is saved and can be read using the Bluetooth APIs. Using the known MAC address for a remote device, a connection can be initiated with it at any time without performing discovery (assuming the device is within range).

Remember there is a difference between being paired and being connected. To be paired means that two devices are aware of each other's existence, have a shared link-key that can be used for authentication, and are capable of establishing an encrypted connection with each other. To be connected means that the devices currently share an RFCOMM channel and are able to transmit data with each other. The current Android Bluetooth API's require devices to be paired before an RFCOMM connection can be established. (Pairing is automatically performed when you initiate an encrypted connection with the Bluetooth APIs.)

The following sections describe how to find devices that have been paired, or discover new devices using device discovery.

Note: Android-powered devices are not discoverable by default. A user can make the device discoverable for a limited time through the system settings, or an application can request that the user enable discoverability without leaving the application. How to enable discoverability is discussed below.

Querying paired devices

Before performing device discovery, its worth querying the set of paired devices to see if the desired device is already known. To do so, call getBondedDevices(). This will return a Set of BluetoothDevices representing paired devices. For example, you can query all paired devices and then show the name of each device to the user, using an ArrayAdapter:

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
// If there are paired devices
if (pairedDevices.size() > 0) {
    // Loop through paired devices
    for (BluetoothDevice device : pairedDevices) {
        // Add the name and address to an array adapter to show in a ListView
        mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
    }
}
```

Figure 9:

Connecting Devices

In order to create a connection between your application on two devices, you must implement both the server-side and client-side mechanisms, because one device must open a server socket and the other one must initiate the connection (using the server device's MAC address to initiate a connection). The server and client are considered connected to each other when they each have a connected BluetoothSocket on the same RFCOMM channel. At this point, each device can obtain input and output streams and data transfer can begin, which is discussed in the section about Managing a Connection. This section describes how to initiate the connection between two devices.

The server device and the client device each obtain the required BluetoothSocket in different ways. The server will receive it when an incoming connection is accepted. The client will receive it when it opens an RFCOMM channel to the server.

One implementation technique is to automatically prepare each device as a server, so that each one has a server socket open and listening for connections. Then either device can initiate a connection with the other and become the client. Alternatively, one device can explicitly "host" the connection and open a server socket on demand and the other device can simply initiate the connection.

Note: If the two devices have not been previously paired, then the Android framework will automatically show a pairing request notification or dialog to the user during the connection procedure, as shown in Figure 3. So when attempting to connect devices, your application does not need to be concerned about whether or not the devices are paired. Your RFCOMM connection attempt will block until the user has successfully paired, or will fail if the user rejects pairing, or if pairing fails or times out.

About UUID

A Universally Unique Identifier (UUID) is a standardized 128-bit format for a string ID used to uniquely identify information. The point of a UUID is that it's big enough that you can select any random and it won't clash. In this case, it's used to uniquely identify your application's Bluetooth service. To get a UUID to use with your application, you can use one of the many random UUID generators on the web, then initialize a UUID with fromString(String).

Connecting as a client

In order to initiate a connection with a remote device (a device holding an open server socket), you must first obtain a BluetoothDevice object that represents the remote device. (Getting a BluetoothDevice is covered in the above section about Finding Devices.) You must then use the BluetoothDevice to acquire a BluetoothSocket and initiate the connection.

Here's the basic procedure:

1. Using the BluetoothDevice, get a BluetoothSocket by calling createRfcommSocketToServiceRecord(UUID).

This initializes a BluetoothSocket that will connect to the BluetoothDevice.

The UUID passed here must match the UUID used by the server device when it opened its BluetoothServerSocket (with listenUsingRfcommWithServiceRecord(String, UUID)). Using the same UUID is simply a matter of hard-coding the UUID string into your application and then referencing it from both the server and client code.

```
private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;

    public ConnectThread(BluetoothDevice device) {
        // Use a temporary object that is later assigned to mmSocket,
        // because mmSocket is final
        BluetoothSocket tmp = null;
        mmDevice = device;

        // Get a BluetoothSocket to connect with the given BluetoothDevice
        try {
            // MY_UUID is the app's UUID string, also used by the server code
            tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) { }
        mmSocket = tmp;
    }

    public void run() {
        // Cancel discovery because it will slow down the connection
        mBluetoothAdapter.cancelDiscovery();

        try {
            // Connect the device through the socket. This will block
            // until it succeeds or throws an exception
            mmSocket.connect();
        } catch (IOException connectException) {
            // Unable to connect; close the socket and get out
            try {
                mmSocket.close();
            } catch (IOException closeException) { }
            return;
        }
```

2.Initiate the connection by calling connect().

Upon this call, the system will perform an SDP lookup on the remote device in order to match the UUID. If the lookup is successful and the remote device accepts the connection, it will share the RFCOMM channel to use during the connection and connect() will return. This method is a blocking call. If, for any reason, the connection fails or the connect() method times out (after about 12 seconds), then it will throw an exception.

Because connect() is a blocking call, this connection procedure should always be performed in a thread separate from the main activity thread.

Note: You should always ensure that the device is not performing device discovery when you call connect(). If discovery is in progress, then the connection attempt will be significantly slowed and is more likely to fail.

Example

Here is a basic example of a thread that initiates a Bluetooth connection:

Notice that cancelDiscovery() is called before the connection is made. You should always do this before connecting and it is safe to call without actually checking whether it is running or not (but if you do want to check, call isDiscovering()).

manageConnectedSocket() is a fictional method in the application that will initiate the thread for transferring data, which is discussed in the section about Managing a Connection.

When you're done with your BluetoothSocket, always call close() to clean up. Doing so will immediately close the connected socket and clean up all internal resources.

How to implement Android Splash Screen

Android splash screen are normally used to show user some kind of progress before the app loads completely. Some people uses splash screen just to show case their app / company logo for a couple of second. Unfortunately in android we don?t have any inbuilt mechanism to show splash screen compared to iOS. In this tutorial we are going to learn how to implement splash screen in your android application.

```
        // Do work to manage the connection (in a separate thread)
        manageConnectedSocket(mmSocket);
    }

    /** Will cancel an in-progress connection, and close the socket */
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) { }
    }
}
```

Figure 10:

1. Create a new project in Eclipse by navigating to File ? New Android ? Application Project and fill required details. (I kept my main activity name as MainActivity.java)

2. For Splash Screen we are creating a separate activity. Create a new class in your package and name it as Splash-Screen.java

3. Open your your AndroidManifest.xml file and make your splash screen activity as Launcher activity.

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="info.androidhive.androidsplashscreentimer"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <!-- Splash screen -->
        <activity
            android:name="info.androidhive.androidsplashscreentimer.SplashScreen"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
            android:theme="@android:style/Theme.Black.NoTitleBar" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- Main activity -->
        <activity
            android:name="info.androidhive.androidsplashscreentimer.MainActivity"
            android:label="@string/app_name" >
        </activity>
    </application>

</manifest>
```

Figure 11:

4. Create a layout file for splash screen under res ? layout folder. I named the layout file as activity_splash.xml. This layout normally contains your app logo or company logo.

ACTIVITY_SPLASH.XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/gradient_background" >

    <ImageView
        android:id="@+id/imgLogo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:src="@drawable/wwe_logo" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="10dp"
        android:textSize="12dp"
        android:textColor="#454545"
        android:gravity="center_horizontal"
        android:layout_alignParentBottom="true"
        android:text="www.androidhive.info" />

</RelativeLayout>
```

Figure 12:

5. Add the following code in SplashScreen.java activity. In this following code a handler is used to wait for specific time and once the timer is out we launched main activity.

SplashScreen.java

```
package info.androidhive.androidsplashscreentimer;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;

public class SplashScreen extends Activity {

    // Splash screen timer
    private static int SPLASH_TIME_OUT = 3000;

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

        new Handler().postDelayed(new Runnable() {

            /*
             * Showing splash screen with a timer. This will be useful when you
             * want to show case your app logo / company
             */

            @Override
            public void run() {
                // This method will be executed once the timer is over
                // Start your app main activity
                Intent i = new Intent(SplashScreen.this, MainActivity.class);
                startActivity(i);

                // close this activity
                finish();
            }
        }, SPLASH_TIME_OUT);
    }
```

Figure 13:

The communication is done with bluetooth module namely BLUELINK.

- BlueLINK is a compact Bluetooth Module (5V Serial TTL) from rhydoLABZ. The module has built-in Voltage regulator and 3V3 to 5V level converter that can be used to interface with 5V Microcontrollers. The module has only 5 pins (Standard 2.54mm berg strip) VCC, GND, TX, RX and RESET. The module is factory configured in Transparent Mode and hence there is no command required for normal operation.
- The BlueLINK is a Drop-in replacement for wired serial connections, transparent usage. You can use it simply for serial port replacement to establish connection between MCU and GPS, PC to your embedded project / Robot etc. Any serial stream from 9600 to 115200 bps can be passed seamlessly from your PC/PDA/MOBILE to your target board!

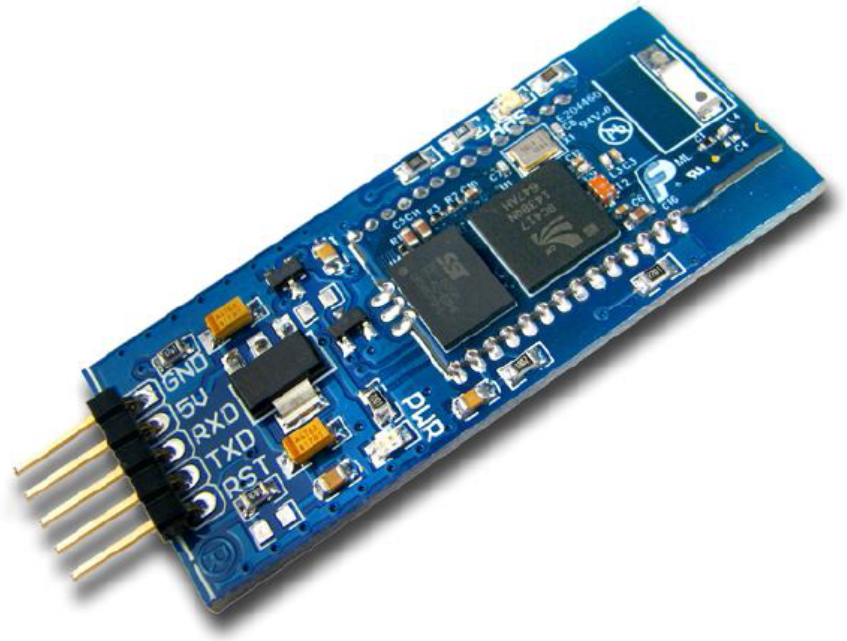

Figure 14: BlueLINK - Bluetooth Module (5V Serial UART)

FEATURES

- Support Master & Slave Mode
- 5-Pin Standard Bergstrip
- Bluetooth core V 2.0 compliant
- SPP (Serial Port Profile) support
- Support UART interface to host system
- Serial communications @ 9600-115200bps
- No Setup/Initial command required
- Breadboard Compatible
- Onboard Status and Power LED
- Encrypted connection
- Frequency: 2.4 to 2.524 GHz
- Built-in Chip antenna
- Power Supply: 5V
- Dimension: 55mm x 19mm x 3.2 mm
- Operating Temperature: -40 to +70C

PIN DEFINITIONS

PIN	PIN NAME	DETAILS
GND	Ground	Ground Level of Power supply
5V	Power Supply	Power Supply Input (5V)
RXD	Receive	Pin for Data Reception
TXD	Transmit	Pin for Data Transmission
RST	Reset	Reset Input (Internally Pulled-Up)

Interfacing of Bluetooth Module

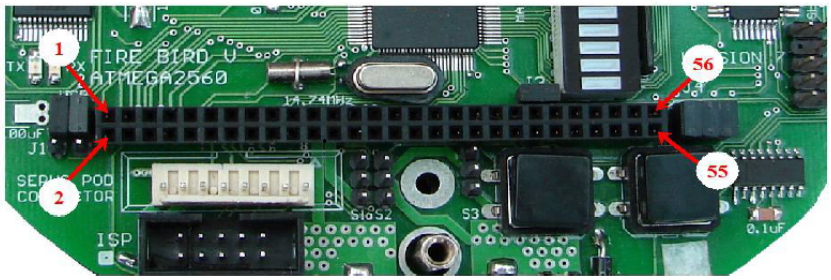

Figure 15: Expansion Socket on ATMEGA 2560 Microcontroller Board

- In interfacing of BLUELINK the connections were made with expansion slot in which the vcc and ground were connected to pin no. 21 and 23 respectively.The Tx of expansion slot was connected to Rx of BLUELINK and vice versa.It is up us to decide that to whcih UART we wish to connect.In Firebird V there are 3 UART available namely UART1,UART2 and UART3.We have connected on UART3.A universal asynchronous receiver/transmitter, abbreviated UART is a piece of computer hardware that translates data between parallel and serial forms.

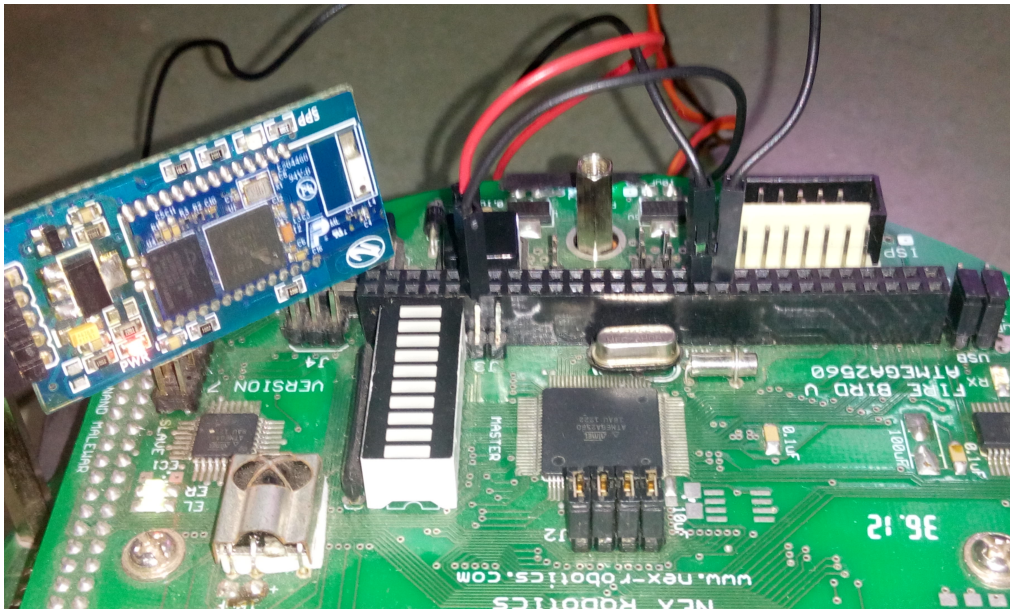

Figure 16: Bluetooth Interfacing on Firebird V Expansion Socket on ATMEGA 2560 Microcontroller Board

- In our robot data is transmitted serially which caused us a lot of problem initially as the UDR register of only 8 bit and the values of velocity and other parameters takes up more than 8 bit.Also since data is being transmitted serially we have to send an identifier with our velocity so that the microcontroller can process it.
- The basic logic of our code is that the data which is send by phone are some special ASCII characters which has some specific meaning associated with it for example: if the button of forward is presseed in our UI then ASCII of 'a' is send and is captured on interupt by our UDR register then simple logic of if else is used to make the robot move forward.Similarly for backward,left,right,buzzer,all the sensors and velocity.In case of sensors ASCII of each sensor is send and then analog values of all sensors is echoed back to our phone.
- The same logic is used for gesture mode in which when our phone is tilted forward the values of accelerometer are changed and accordingly the data is send to bot to move forward.Similar logic is defined for all other directions as well.We resolved the constraint of 8 bit memory by breaking the data and then sending it.

Bugs:

- The major problem with bluetooth module is that as it is short range it takes time to connect with bluetooth of our phone.We should keep our phone nearer to it so that the connection can be established faster.One major problem is that if once the connection is lost then robot has to be switched off and turned on again in order to establish connection again.
- Also another problem with the app is that when you enter the gesture mode then sometimes it causes some unknown problem and the app crashes.There is still some need for the calibration of the readings of the accelerometer.

- One more issue is that there is one second delay between the updated values of the sensors.The rest of the app works perfectly fine.

Future Work: In future work the same project can be implemented with wifi module and USB cable instead of bluetooth module.The possibility of improving the UI is always there.

References:

- www.stackoverflow.com
- www.developer.android.com
- www.androidhive.info
- www.tutorialspoint.com