Wheel Balance and Alignment Sensor

Nicholas McNally, Computer Engineering

Project Advisor: Dr. Marc Mitchell

April 22, 2016
Evansville, Indiana
Acknowledgements

I would like to thank Dave McNally for supplying materials and advice in construction of the testing unit.
# Table of Contents

I. Introduction  

II. Background  

III. Design Approach  
   A. Hardware Design  
   B. Software Design  
   C. Considerations and Constraints  
   D. Implementation  
   E. Costs  

IV. Results  

V. Conclusion  

VI. References
Table of Figures

1. Wheel Balancing Machine
2. Wheel Spacer Attached to Wheel Hub
3. Mock Wheel
4. Testing unit to hold mock wheel
5. Triple Axis Accelerometer
6. RFduino BLE microcontroller
7. RFduino USB programming shield
8. Possible Alignment Issues
9. Screenshot of Android application

Tables

1. Costs of components in creation of this project
I. Introduction

Vehicles these days are technologically advanced, yet they do not have any systems for informing the driver if the wheels are off balance or out of alignment. For this reason a device has been designed that attaches to the wheel of a vehicle and notifies the user of any issues regarding wheel balance or alignment. Through the app on the user’s smart phone, they are notified of the issues so they may attend to the issues sooner than regularly expected as to prevent any lasting damage to the vehicle or themselves.

II. Background

An unbalanced or out of alignment wheel can cause vehicle handling problems, notably, a shaking steering wheel. In this day and age, many vehicle operators are not knowledgeable as to when their vehicle is not performing correctly. If the problem is not corrected in a timely manner, more parts of the vehicle could become dislodged from the shaking, or an accident could occur due to inhibited vehicle handling. There is currently no item on the market that provides these services to the driver. Shops have machines to detect the issues (Figure 1). To use this machine, a mechanic will secure the wheel and tire onto the axel of the balancing machine and the machine will spin the wheel, measuring where the wheel needs weight and how much to add. However these machines are too
large and expensive for the average vehicle owner to consider purchasing, especially when the average driver would use it only once or twice a year.

Client requirements are as follows:

- Needs to be able to sense if the wheel the device is attached to is balanced.
- Must be able to alert user via an application on an Android device of the wheel status.

III. Design Approach

The final design of the device is small enough to fit between the wheel and the brake, much like a wheel spacer in Figure 2. A wheel spacer is generally used to allow the owner of a vehicle to fit wider wheels on the car so that the new wheels will not scrape the interior of the wheel wells. The idea to fashion the device as a wheel spacer provided a base from which to begin generalizing device design to allow the device to be close enough to the wheel in order to obtain accurate measurements. In the final design of this project, the mock wheel does resemble a thick wheel spacer, although it is not capable of being mounted on the wheel of a vehicle due to physical constraints and the safety hazards that would incur.

A. Hardware Design
Due to physical constraints, a mock wheel (Figure 3) containing the measurement components is attached to a mounted motor (Figure 4) that rotates the mock wheel in order to obtain balance and alignment data. The components used to attain the measurements are as follows:

- **Triple Axis Accelerometer  Gyro Breakout MPU-6050** (Figure 5)
  - This is used for measurement of accelerations in three dimensions as well as the wheel’s position at the time of the measurement.

- **Rfduino Bluetooth low energy (BLE) microcontroller** (Figure 6)
  - This is used for collecting the data from the accelerometer, interpreting the collected data, and informing the application of the results.

- **Rfduino USB programming shield** (Figure 7)
  - This device plugs directly into the RFduino to allow the microcontroller code to be loaded and modified.

- **Google Nexus 7 Tablet**
Used for application testing purposes however, any Android device running at least version 4.3 is capable of using the application.

- 3.3v button cell battery
  - Used to power measurement devices on the mock wheel.
- 3D printed mock wheel (Figure 3)
  - Used to contain measurement components
- Wheel testing unit (Figure 4)
  - A mounted motor capable of rotating the mock wheel for small scale testing.

The mock wheel contains a button cell battery to power an accelerometer and the RFduino. When the RFduino receives a start command from the application, it will wake the accelerometer from its sleep mode and begin collecting data via I2C as well as interpreting acceleration data.

The schematics for the hardware components can be found in appendix A

### B. Software Design

The application running on the Nexus tablet was designed in Android Studio [9]. Android Studio is an integrated development environment made by Android developers and is open-source to encourage people to build their own applications.

The application is built to connect to the testing device and display the balance and alignment results that are calculated on the RFduino. When the application is started, it checks to make sure that the measurement device is connected to the Android device. If the measurement device is not connected, the user is prompted to connect before continuing. After
connecting, the user is informed to press start after they are already driving down as smooth and straight of a road as possible. The road must be as smooth as possible in order to prevent collection of bad data due to a bumpy road that would be difficult for the RFduino to interpret.

After data collection has begun, the accelerometer collects data in the form of accelerometer readings. The accelerometer data tells how much acceleration in the x, y, and z is occurring at that moment of data collection. The goal of the data collection is to build a map of accelerations through one rotation of the wheel. In order to index this map, the RFduino looks at the angle theta between the x and y acceleration with respect to the x-axis. The map is indexed to every five degrees in order to save time on data collection while still acquiring enough data to make an informed decision. At the index calculated, the vector magnitude of the x and y acceleration components is placed for later reference to infer the state of the wheel balance. At the same index in another array, the z acceleration is placed for reference in calculation of the alignment state. Once the full rotation of the wheel has been mapped with acceleration data, the data collection is stopped.

Now that the RFduino has composed a map of accelerations thorough a rotation it interprets the data into useful information regarding wheel balance and alignment. In order to know if the wheel is statically balanced, the difference between the magnitude at the top of the wheel rotation and the bottom of the rotation should equal zero. The same should hold true for
all magnitudes and their 180-degree counter-parts. Regarding wheel alignment, with only the proposed device attached to a wheel, the only alignment issue that can be detected is camber as described in Figure 8. This is due to the fact that testing other forms of alignment require external testing equipment that is not practically possible to mount to a vehicle. Camber is how the wheel is positioned vertically towards the inside or outside of the vehicle [10], ideally, there should be no camber and the wheel should be vertical with no lean, this is where the values on the z axis from the accelerometer come into play. A wheel that has a positive or negative camber will show a changing z value around the upper and lower portions of the wheel rotation map. Once all calculations are complete, the RFduino passes the results back to the application (Figure 9) and the application then notifies the user of the current balance and alignment state of their wheel.

The complete code for this software can be found in appendix B.

C. Considerations and Constraints

Relevant IEEE standards to be met include:

• 1633-2008 - IEEE Recommended Practice on Software Reliability, methods of assessing and assuring consistently reliable software [7]

802.15.1-2005 - IEEE Standard for Information technology is necessary because of the use of the Bluetooth modem used in the measuring device.

1633-2008 - IEEE Recommended Practice on Software Reliability in relevant in the production of the android application with which the measuring device communicates with.

This product is cheap to produce and with the possible profit margin, this product is economically viable. Most of the mass of the product can be made from recycled metal, making it environmentally friendly. This product does not make any difference in social, political, or ethical communities.

As a final product that is in use on a vehicle inside of a metal shell safety issues include:

The device is too wide, preventing the lug nuts that attach the wheel to the hub from tightening enough to keep the wheel attached. Being that this device is making physical contact with the brake disk, there is a risk of the circuit components catching fire from heat caused by the brake disks. If the device shell is not robust enough and were to contort under an extreme pressure, causing the wheel to immediately become unpredictable while the vehicle is at speed. Correcting the aforementioned safety issues is beyond the scope of knowledge of personnel tasked with this problem.

D. Implementation
In order to test this device, the measuring components are be placed in the center cavity of a mock ‘wheel’ as seen in figure 7. The mock wheel rotates about a shaft through the center connected to a fan motor mounted on a sturdy base as seen in figure 8. In order to test the balancing capabilities, small weights, such as coins, are taped to the outer rim of the mock wheel in order to simulate an imbalance. As for testing the camber detection capabilities, the wheel is simply be given a vertical lean. The lean is accomplished by turning the bolt on the back of the base, tilting the mock wheel to simulate a non-zero degree camber.

\[E.\ Costs\]

Table 1: Costs of components in creation of this project

<table>
<thead>
<tr>
<th>Item:</th>
<th>Cost:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RioRand(TM) Triple Axis Accelerometer&amp;Gyro Breakout MPU-6050</td>
<td>$10.95</td>
</tr>
<tr>
<td>Rfuino</td>
<td>$30</td>
</tr>
<tr>
<td>3Dprinted wheel</td>
<td>$10</td>
</tr>
<tr>
<td>Android device</td>
<td>Provided</td>
</tr>
<tr>
<td>Total</td>
<td>$50.95</td>
</tr>
</tbody>
</table>
IV. Results

The outcome of this project is that the user is able to connect to the measuring device attached to the mock wheel via an Android device application capable of handling the Bluetooth connection and displaying data it receives. The RFduino takes acceleration measurements and places them into their correct indices to build the wheel rotation map. However, the RFduino has difficulty collecting enough data to make an inference about the wheel balance and alignment. The application designed to interface with the testing device does not compile therefore the product presentation was completed on the aforementioned application. The research done in this project will be very useful in developing a device much like a wheel spacer built to fit on actual vehicles, or possibly integrated into new vehicles at the factory as some component in the wheel that is connected to the vehicle’s computer.

V. Conclusion

This project proves that it is possible to monitor the wheel balance and basic alignment characteristics on a moving wheel. With collaboration with technicians that know more about manufacturing parts that can stand up to the abuse of being attached to the wheel of a moving car, this idea can realistically become a viable product to be sold as and aftermarket add-on or be manufactured on new vehicles in the factory. If this route is taken, it will save users money that would be wasted on tires, suspension components, or insurance claims, due to a poorly balance or aligned wheel.
Appendix A: Hardware

A-1: Wheel balance and alignment sensing hardware schematic

[Diagram of wheel balance and alignment sensing hardware schematic]

A-2: Testing wheel schematic

[Diagram of testing wheel schematic]
Appendix B: Software

B-1: RFduino code

#include<Wire.h>
#include<RFduinoBLE.h>
#include<math.h>

//================================Function declarations=================================

double square(double input);
boolean dataCollected();
void checkBalance();

//=================================Global variables=================================
const int MPU_addr = 0x68; // I2C address of the MPU-6050
int16_t AcX, AcY, AcZ;
double rotationPos[36] = {0};
double rotationNeg[36] = {0};
bool start = false;

void setup() {
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);  // PWR_MGMT_1 register
  Wire.write(0);     // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x1C);
  Wire.write(0xff);
  Wire.endTransmission(true);
  Serial.begin(9600);
  RFduinoBLE.begin();
}

void RFduinoBLE_onConnect() {
  start = true;
}

void RFduinoBLE_onDisconnect()
{
    start = false;
}

void loop() {
    if (start) {
        Wire.beginTransmission(MPU_addr);
        Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
        Wire.endTransmission(false);
        Wire.requestFrom(MPU_addr, 14, true); // request a total of 14 registers
        AcX = Wire.read() << 8 | Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
        AcY = Wire.read() << 8 | Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
        AcZ = Wire.read() << 8 | Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
        double f = sqrt(square(AcX) + square(AcY));
        double theta = (atan(AcX / AcY) * 4068) / 71;
        int ind = abs(round(theta / 5));
        RFduinoBLE.send(f);
        RFduinoBLE.send(theta);
        if (AcY > 0) {
            if (rotationPos[ind] != 0)
                rotationPos[ind] = (rotationPos[ind] + f) / 2;
            else
                rotationPos[ind] = f;
        }
    }
}
else {
    if (rotationNeg[ind] != 0)
        rotationNeg[ind] = (rotationNeg[ind] + f) / 2;
    else
        rotationNeg[ind] = f;
} //end if-else

if (dataCollected()) {
    start = false;
    checkBalance();
}
}
delay(500);

//=================Helper Functions======================
double square(double input) {
    return (input * input);
}

boolean dataCollected() {
    int i;
    for (i = 0; i < 36; i++) {
        if (rotationPos[i] == 0 || rotationNeg[i] == 0)
            return false;
    }
    return true;
}

void checkBalance() {
}
```java
int i;
for (i = 0; i < 36; i++) {
    if (rotationPos[i] != rotationNeg[i]) {
        RFduinoBLE.sendInt(0);
        return;
    }
}
RFduinoBLE.sendInt(1);

B-2: MainActivity.java
package com.example.nick.wheel_balance_alignment;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.os.IBinder;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity implements
    BluetoothAdapter.LeScanCallback {

    //variables
    final private static int STATE_BLUETOOTH_OFF = 1;
    final private static int STATE_DISCONNECTED = 2;
    final private static int STATE_CONNECTING = 3;
    final private static int STATE_CONNECTED = 4;

    private int currState;
```
private boolean scanStarted, scanning;

private BluetoothAdapter BTadapter;
private BluetoothDevice BTdevice;

private RFduinoService RFDservice;

private Button button_start;
private TextView balanceState;
private TextView alignmentState;

private final BroadcastReceiver BTstateReveiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        int state = intent.getIntExtra(BTadapter.EXTRA_STATE, 0);
        if (state == BTadapter.STATE_ON) {
            upgradeState(STATE_DISCONNECTED);
        } else if (state == BTadapter.STATE_OFF) {
            downgradeState(STATE_BLUETOOTH_OFF);
        }
    }
};

private final BroadcastReceiver scanModeReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        scanning = BTadapter.getScanMode() != BTadapter.SCAN_MODE_NONE;
        scanStarted &= scanning;
        updateUi();
    }
};

private final ServiceConnection RFDserviceConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        RFDservice = ((RFduinoService.LocalBinder) service).getService();
        if (RFDservice.init()) {
            if (RFDservice.connect(BTdevice.getAddress())) {
                upgradeState(STATE_CONNECTING);
            }
        }
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
        RFDservice = null;
        downgradeState(STATE_DISCONNECTED);
    }
};
private final BroadcastReceiver RFDreceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        if(RFduinoService.ACTION_CONNECTED.equals(action))
            upgradeState(STATE_CONNECTED);
        else if(RFduinoService.ACTION_DISCONNECTED.equals(action))
            downgradeState(STATE_DISCONNECTED);
        else if(RFduinoService.ACTION_DATA_AVAILABLE.equals(action))
            addData(intent.getByteArrayExtra(RFduinoService.EXTRA_DATA));
    }
};

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

    BTadapter = BluetoothAdapter.getDefaultAdapter();

    //Bluetooth
    //TODO start button implementation possibly consisting of BT and find device

    @Override
    protected void onStart() {
        super.onStart();
        registerReceiver(scanModeReceiver, new IntentFilter(BluetoothAdapter.ACTION_SCAN_MODE_CHANGED));
        registerReceiver(BTstateReveiver, new IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED));
        registerReceiver(RFDreceiver, RFduinoService.getActionFilter());
        updateState(BTadapter.isEnabled() ? STATE_DISCONNECTED : (BTadapter.isEnabled() ? STATE_BLUETOOTH_OFF));
    }

    @Override
    protected void onStop() {
        super.onStop();
        BTadapter.stopLeScan(this);
        unregisterReceiver(scanModeReceiver);
        unregisterReceiver(BTstateReveiver);
    }
}
unregisterReceiver(RFDreceiver);
}

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

//noinspection SimplifiableIfStatement
if (id == R.id.action_settings) {
    return true;
}

return super.onOptionsItemSelected(item);

@Override
public void onLeScan(BluetoothDevice device, final int rssi, final byte[] scanRecord) {
    Log.d("ending LeScan", "before stop");
    BTadapter.stopLeScan(this);
    BTdevice = device;
    Log.d("onLeScan", "on LeScan ... close previous scan and start scan");
    MainActivity.this.runOnUiThread(new Runnable() {
        @Override
        public void run() {
            deviceInfoText.setText(BluetoothHelper.getDeviceInfoText(BTdevice, rssi, scanRecord));
            Log.d("inside of run", "device is =" + BTdevice + "rssi" + rssi + "and scanRecord is " + scanRecord);
            updateUi();
        }
    });
}

private void upgradeState(int state){
    if(state>currState)
updateState(state);
}

private void downgradeState(int state){
    if(state<currState)
        updateState(state);
}

private void updateState(int state){
    currState = state;
    updateUi();
}

private void updateUi(){
    //TODO
}

private void addData(byte[] data){
    //TODO
}

//TODO

B-3: BluetoothHelper.java

package com.example.nick.wheel_balance_alignment;

import android.bluetooth.BluetoothDevice;
import java.util.UUID;

public class BluetoothHelper {
    public static String shortUuidFormat = "0000%04X-0000-1000-8000-00805F9B34FB";

    public static UUID sixteenBitUuid(long shortUuid){
        assert shortUuid >= 0 && shortUuid<= 0xFFFF;
        return UUID.fromString(String.format(shortUuidFormat, shortUuid & 0xFFFF));
    }

    public static String getDeviceInfoText(BluetoothDevice device, int rssi, byte[] scanRecord){
        return new StringBuilder()
            .append("Name: ").append(device.getName())
            .append("nMAC: ").append(device.getAddress())
            .append("nRSSI: ").append(rssi)
private static String parseScanRecord(byte[] scanRecord) {
    StringBuilder retval = new StringBuilder();
    int i = 0;
    while(i<scanRecord.length) {
        int len = scanRecord[i++]&0xFFFF;
        if(len == 0)
            break;
        switch(scanRecord[i]&0xFFFF) {
            case 0x0A: // TX power
                retval.append("\n TX power: ").append(scanRecord[i++]);
                break;
            case 0xFF: // Specific data
                retval.append("\n Advertisement Data: ")
                    .append(HexAsciiHelper.bytesToHex(scanRecord, i + 3, len));
                String ascii = HexAsciiHelper.bytesToAsciiMaybe(scanRecord, i+3, len);
                if(ascii != null)
                    retval.append(" (").append(ascii).append(")");
                break;
        } 
        i += len;
    }
    return retval.toString();
}

B-4: HexAsciiHelper.java

package com.example.nick.wheel_balance_alignment;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;
import org.apache.http.util.ByteArrayBuffer;
import java.util.regex.Pattern;

public class HexAsciiHelper {
    // Method implementation...
}
public static int PRINTABLE_ASCII_MIN = 0x20; // ' '
public static int PRINTABLE_ASCII_MAX = 0x7E; // '~'

public static boolean isPrintableAscii(int c) {
    return c >= PRINTABLE_ASCII_MIN && c <= PRINTABLE_ASCII_MAX;
}

public static String bytesToHex(byte[] data) {
    return bytesToHex(data, 0, data.length);
}

public static String bytesToHex(byte[] data, int offset, int length) {
    if (length <= 0) {
        return "";
    }

    StringBuilder hex = new StringBuilder();
    for (int i = offset; i < offset + length; i++) {
        hex.append(String.format(" %02X", data[i] % 0xFF));
    }
    hex.deleteCharAt(0);
    return hex.toString();
}

public static String bytesToAsciiMaybe(byte[] data) {
    return bytesToAsciiMaybe(data, 0, data.length);
}

public static String bytesToAsciiMaybe(byte[] data, int offset, int length) {
    StringBuilder ascii = new StringBuilder();
    boolean zeros = false;
    for (int i = offset; i < offset + length; i++) {
        int c = data[i] & 0xFF;
        if (isPrintableAscii(c)) {
            if (zeros) {
                return null;
            }
            ascii.append((char) c);
        } else if (c == 0) {
            zeros = true;
        } else {
            return null;
        }
    }
    return ascii.toString();
}
public static byte[] hexToBytes(String hex) {
    ByteArrayOutputStream bytes = new ByteArrayOutputStream(hex.length() / 2);
    for (int i = 0; i < hex.length(); i++) {
        if (hex.charAt(i) == ' ') {
            continue;
        }

        String hexByte;
        if (i + 1 < hex.length()) {
            hexByte = hex.substring(i, i + 2).trim();
            i++;
        } else {
            hexByte = hex.substring(i, i + 1);
        }

        bytes.append(Integer.parseInt(hexByte, 16));
    }

    return bytes.toByteArray();
}

B-5: RFduinoService.java

package com.example.nick.wheel_balance_alignment;

import android.Manifest;
import android.app.Service;
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.bluetooth.BluetoothGattService;
import android.bluetooth.BluetoothManager;
import android.content.Context;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;
import java.util.UUID;
public class RFduinoService {

    private final static String TAG = RFduinoService.class.getSimpleName();

    private final IBinder mBinder = new LocalBinder();

    private BluetoothManager BTmanager;
    private BluetoothAdapter BTadapter;
    private String BTdeviceAddr;
    private BluetoothGatt BTgatt;
    private BluetoothGattService BTgattService;

    public final static String ACTION_CONNECTED = "com.rfduino.ACTION_CONNECTED";
    public final static String ACTION_DISCONNECTED = "com.rfduino.DISCONNECTED";
    public final static String ACTION_DATA_AVAILABLE = "com.rfduino.ACTION_DATA_AVAILABLE";
    public final static String ACTION_EXTRA_DATA = "com.rfduino.ACTION_EXTRA_DATA";

    public final static UUID UUID_SERVICE = BluetoothHelper.sixteenBitUuid(0x2220);
    public final static UUID UUID_RECEIVE = BluetoothHelper.sixteenBitUuid(0x02221);
    public final static UUID UUID_SEND = BluetoothHelper.sixteenBitUuid(0x2222);
    public final static UUID UUID_DISCONNECT = BluetoothHelper.sixteenBitUuid(0x2223);
    public final static UUID UUID_CLIENT_CONFIGURATION = BluetoothHelper.sixteenBitUuid(0x2902);
    public final static String EXTRA_DATA = "com.rfduino.EXTRA_DATA";

    //Implements callback methods for GATT events that we care about
    private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {
        @Override
        public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
            if(newState == BluetoothProfile.STATE_CONNECTED){
                Log.i(TAG, "Connected to RFduino.");
                Log.i(TAG, "Attemptin to start service discovery: " + BTgatt.discoverServices());
            }
            else if(newState == BluetoothProfile.STATE_DISCONNECTED){
                Log.i(TAG, "Disconnected from RFduino.");
                broadcastUpdate(ACTION_DISCONNECTED);
            }
        }

        @Override
        public void onServicesDiscovered(BluetoothGatt gatt, int status){
            if(status == BluetoothGatt.GATT_SUCCESS){
                BTgattService = gatt.getService(UUID_SERVICE);
            }
        }
    }
if(BTgattService == null){
    Log.e(TAG, "RFduino GATT service not found.");
    return;
}
BluetoothGattCharacteristic receiveCharacteristic =
    BTgattService.getCharacteristic(UUID_RECEIVE);
if(receiveCharacteristic != null){
    BluetoothGattDescriptor receiveConfigDescriptor =
        receiveCharacteristic.getDescriptor(UUID_CLIENT_CONFIGURATION);
    if(receiveConfigDescriptor != null){
        gatt.setCharacteristicNotification(receiveCharacteristic, true);
        receiveConfigDescriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
        gatt.writeDescriptor(receiveConfigDescriptor);
    }
    else
        Log.e(TAG, "RFduino receive config descriptor not found.");
}
else
    Log.e(TAG, "RFduino receive characteristics not found.");

    broadcastUpdate(ACTION_CONNECTED);
}
else
    Log.v(TAG, "onServicesDiscovered received: " + status);
}

@Override
public void onCharacteristicRead(BluetoothGatt gatt, BluetoothGattCharacteristic chara, int status){
    if(status == BluetoothGatt.GATT_SUCCESS)
        broadcastUpdate(ACTION_DATA_AVAILABLE, chara);
}

@Override
public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic chara){
    broadcastUpdate(ACTION_DATA_AVAILABLE, chara);
}

private void broadcastUpdate(final String action) {
    final Intent intent = new Intent(action);
sendBroadcast(intent, android.Manifest.permission.BLUETOOTH);

private void broadcastUpdate(final String action, final BluetoothGattCharacteristic chara) {
    if (UUID_RECEIVE.equals(chara.getUuid())){
        final Intent intent = new Intent(action);
        intent.putExtra(EXTRA_DATA, chara.getValue());
        sendBroadcast(intent, android.Manifest.permission.BLUETOOTH);
    }
}

public class LocalBinder extends Binder{
    RFduinoService getService(){
        return RFduinoService.this;
    }
}

@Override
public IBinder onBind(Intent intent) {
    return mBinder;
}

@Override
public boolean onUnbind(Intent intent){
    close();
    return super.onUnbind(intent);
}

public boolean init(){
    if(BTmanager == null){
        BTmanager = (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
        if(BTmanager == null){
            Log.e(TAG, "Unable to initialize BluetoothManager");
            return false;
        }
    }
    BTadapter = BluetoothManager.getAdapter();
    if(BTadapter == null){
        Log.e(TAG, "Unable to obtain a BluetoothAdapter");
        return false;
    }
    return true;
}

//connect to the GATT server
public boolean connect(final String addr){

if(BTadapter == null || addr == null) {
    Log.v(TAG, "BluetoothAdapter not initialized or unspecified address");
    return false;
}

// try to connect to previously connected device
if(BTdeviceAddr != null && addr.equals(BTdeviceAddr) && BTgatt != null) {
    Log.d(TAG, "Trying to use an existing BTgatt for connection");
    return BTgatt.connect();
}

final BluetoothDevice device = BTadapter.getRemoteDevice(addr);
BTgatt = device.connectGatt(this, false, mGattCallback);
Log.d(TAG, "Trying to create a new connection");
BTdeviceAddr = addr;
return true;

// disconnect from GATT server
public void disconnect() {
    if(BTadapter == null || BTgatt == null) {
        Log.v(TAG, "BluetoothAdapter not initialized");
        return;
    }
    BTgatt.disconnect();
}

// acquiring and releasing GATT resources
public void close() {
    if(BTgatt == null)
        return;
    BTgatt.close();
    BTgatt = null;
}

public void read() {
    if(BTgatt == null || BTgattService == null) {
        Log.v(TAG, "BluetoothGatt not initialized");
        return;
    }
    BluetoothGattCharacteristic chara = BTgattService.getCharacteristic(UUID_RECEIVE);
    BTgatt.readCharacteristic(chara);
}

public boolean send(byte[] data) {
    if(BTgatt == null || BTgattService == null) {

Log.v(TAG,"BluetoothGatt not initialized");
return false;
}
BluetoothGattCharacteristic chara = BTgattService.getCharacteristic(UUID_SEND);
if(chara == null){
    Log.v(TAG, "Send characteristic not found");
    return false;
}
chara.setValue(data);
chara.setWriteType(BluetoothGattCharacteristic.WRITE_TYPE_NO_RESPONSE);
return BTgatt.writeCharacteristic(chara);
}

public static IntentFilter getIntentFilter(){
    IntentFilter filter = new IntentFilter();
    filter.addAction(ACTION_CONNECTED);
    filter.addAction(ACTION_DISCONNECTED);
    filter.addAction(ACTION_DATA_AVAILABLE);
    return filter;
}

B-6: activity_main.xml

<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=".MainActivity"
    android:id="@+id/relativeLayut">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/introduction"
        android:id="@+id/textView_instruction"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
android:text="Start Measurements"
android:id="@+id/startButton"
android:layout_below="@+id/textView_instruction"
android:layout_centerHorizontal="true"
android:layout_marginTop="32dp" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="@android:attr/textAppearanceLarge"
android:text="@string/wheelBalance"
android:id="@+id/textView_wheelBalance"
android:layout_alignTop="@+id/textView_balanceState"
android:layout_alignParentStart="true" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="@android:attr/textAppearanceLarge"
android:text="@string/balanceState_Unknown"
android:id="@+id/textView_balanceState"
android:layout_above="@+id/textView_alignmentState"
android:layout_alignParentEnd="true"
android:layout_marginBottom="43dp" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="@android:attr/textAppearanceLarge"
android:text="@string/wheelAlignment"
android:id="@+id/textView_WheelAlignment"
android:layout_alignParentBottom="true"
android:layout_alignParentStart="true" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="@android:attr/textAppearanceLarge"
android:text="@string/alignmentState_Unknown"
android:id="@+id/textView_alignmentState"
android:layout_alignTop="@+id/textView_WheelAlignment"
android:layout_alignParentBottom="true"
android:layout_alignParentStart="true" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="@android:attr/textAppearanceLarge"
android:text="@string/alignmentState_Unknown"
android:id="@+id/textView_alignmentState"
android:layout_alignTop="@+id/textView_WheelAlignment"
android:layout_alignStart="true" />

</RelativeLayout>
**References**


Available: http://www.amazon.com/gp/product/B00H1OYE4Q?psc=1&redirect=true&ref_=oh_aui_detailpage_o05_s00


[8] IEEE Recommended Practice on Software Reliability, 1633-2008


Available: http://autorepair.about.com/od/glossary/ss/def_camber.htm