Wireless ECG Monitoring

Linzhe Li
Runze Dong
Tian Tian

Faculty Advisor: Dr. Tao Zhang, Dr. Michael Colef
05/16/2014
1. Background Information

1.1 Introduction

ECG (Electrocardiogram) is a signal that is used by medical professionals to determine what is wrong with the heart and the body. Based on medical usage of ECG, we found that the ECG is an extremely important vital signal that needs to be monitored for heart patients.

ECG is a diagnostic procedure that measures electrical activity of the heart muscle over time. What’s more, Medical professionals to determine what is wrong with the heart and body also use ECG signal. The electrical waves can be measured at selectively placed electrodes (electrical contacts) in several points of the body on the skin. Electrodes on different sides of the heart measure the activity of different parts of the heart muscle. An ECG displays the voltage between pairs of these electrodes, and the muscle activity that they measure, from different directions by a specific instrument to show the result in the form of a graph clearly.

A normal heartbeat cycle includes a P wave, a T wave and a QRS complex. We can see the waveform in the following diagram. It is composed of the relationship between the different parts of the waveform can be used to determine the heart rate and distinguish between the various arrhythmias, including
tachycardia (abnormally high heart rhythm > 100 BPM, and bradycardia (abnormally slow heart rhythm <60 BPM).

Fig.1. ECG Signal

1.2 Statement of Problem

The leading killer of Americans in the 21st Century continues to be a non-infectious disease is heart disease. It is the leading reason of the death for both men and women. Each year, about 600,000 people die because of heart disease in the USA. That means there is one person dies of heart disease in every four deaths in America.

It is a great method to continuously monitor ECG for those cardiac patients to prevent them from heart diseases, because timely treatment is the best way for saving their lives. However, we can only find the ECG monitoring machines in hospital. Commonly, they are large, heavy and untransportable which means it
cannot be a long-term measuring system for most of patients. On the one hand, those machines need power from electrical outlets. They are heavy and immovable which result in the inconveniences for patients. On the other hand, the wires which link machine and the sensors together will take hospitals a lot of time space to reserve.

It is important for patients to have a wireless and portable monitoring system, which can be suitable for any person to use. Medical alarming systems will allow the monitoring of a patient and react knowing when a patient is in need of medical attention – often times, long before the patient knows he or she is in need of help. By developing a small, wireless monitoring system that is adaptable for any person to use, it is possible to create systems of communication for multiple patients and their doctors. These devices will be able to communicate wirelessly to nearby cell phones or computers in order to share the data in a way that makes it easy to access for both the patient and their doctor. The ideal system will have in place methods that will help to eliminate errors and expand efficiency of data transmission. Relaying this data back to a cell phone or base computer will allow for further processing of data, data storage, and the ability for data mining. Most importantly, the ability to deliver this data to a doctor’s smartphone for remote monitoring is a key component. When the data contains “life or death” information, speed and efficiency cannot be taken for granted. This technology is generalized enough have multiple applications. The monitoring of patients is
useful for patients at home, patients in nursing home, and those in rehabilitation centers. The patient does not require full attention, as in an ICU, but is not stable enough to go completely unattended. This technology will change the medical field. The same way that Facebook took over social interaction in the 21st Century, medical wireless monitoring will change the way patients are cared for. The road to the preservation of human life in a world full of technology begins here. This paper will explain the steps on how we researched and applied this technology.

2. Statement of Purpose

2.1 Objective

The goal of our project is to design a system that can make an easy way for patients to get their ECG in their smart phones. Doctors can also get an alarm by their smart phones while the patients’ ECG is abnormal. They can take steps immediately, which can prevent patients from death in most case. By using the wireless monitoring system, we can measure the ECG when people are doing some special mission such as spaceflight or diving.

The ideal system will have in place methods that will help to eliminate errors and expand efficiency of data transmission. Relaying this data back to a cell phone or base computer will allow for further processing of data, data storage, and
the ability for data mining. Most importantly, the ability to deliver this data to a doctor’s smartphone for remote monitoring is a key component. When the data contains “life or death” information, speed and efficiency cannot be taken for granted. This technology is generalized enough have multiple applications. The monitoring of patients is useful for patients at home, patients in nursing home, and those in rehabilitation centers.

### 2.2 Plan of Action

For the application of ECG monitoring, we decided to use the Arduino board to achieve data processing and communication functions. The flow of logic for this project has been as such:

1. Take in an ECG signal from the body.
2. Send the ECG signal through a prefix amplification circuit.
3. Pass the low amplified signal to band-pass filter circuit.
4. Amplify the ECG signal in the main amplification circuit.
5. Read amplified signal into Arduino board.
6. Measure voltage and time interval for each ECG pulse.
5. Transmit data via Bluetooth to a smartphone.
6. Compare those data with rated value in smartphone.
7. Send alarm message if irregular signal is found.

Each of these steps in the project has many sub-steps in them, including
hardware, software, and communication capability.

3. Circuits Design

3.1 Flow chart

![Flow Chart](image1.png)

Fig. 2. Flow Chart

3.2 ECG Circuit Architecture

![ECG Circuit](image2.png)

Fig. 3. ECG Circuit
3.3 Amplifier Circuits

3.3.1 Preamplifier Circuit

Preamplifier circuit is the first stage amplifier circuit. Since the amplitude of the input ECG signal is very small and there are interference signals, the main function of preamplifier circuit is to filter out common mode interference and enlarge ECG signal in a limited range. In preamplifier circuit, we choose AD620 as its amplifier. The AD620 is a low cost, high accuracy instrumentation amplifier which requires only one external resistor to set gains of 1 to 10,000[2].

The gain equation is:

\[ G = \frac{49.4k\Omega}{R_G} + 1 \]

We set 10 times at the preamplifier circuits, so that \( R_G \) is approximately equal to 5.49k\( \Omega \).

3.3.2 Post Amplifier Circuit

Post amplifier circuit, which is main amplifier circuit. The ECG signal in itself has a voltage of 1mV ~ 4mV. Due to the positive and negative 5V power supply to amplifier, the output signal cannot exceed the range. What’s more, the Arduino board only can work under 5V. We decide multiple 100 times after the first amplification. That is 1000 times in total.
HA17741 is general-purpose operational amplifier, which gain equation is:

\[ G = \frac{R_G}{R_L} \]

### 3.4 Filter Circuits

Electronic filters are analog circuits, which perform signal-processing functions, specifically to remove unwanted frequency components from the signal, to enhance wanted ones, or both [1]. In generally, we can distinguish filters into two deferent types. Filters are based on combinations of resisters (R), capacitors (C) and inductors (L), known as passive filter.

#### 3.4.1 High-pass Filter

High-pass filter, as the name suggests, allows all frequencies pass, which is above a certain frequency, attenuating low frequency signal and getting low SNR signal. In our design, we utilize passive filter as our high-pass filter and select \( R = 1.5\,\Omega \) and \( C = 10\,\mu F \). In the ideological circumstance, we can calculate its cut-off frequency following the formula below:

\[ f = \frac{1}{2\pi RC} \]

The results should be approximate to 10mHz. But, as we all know, filters frequency response will never sharply drop to 0 in reality. Figure 1 shows the real circumstance, in which we use first-order filter as example. The signal is
attenuating at a rate of 15 dB per decade.

Fig. 4. Frequency response with 15 dB per decade

We connect two first-order high-pass filters in series to get one second-order high-pass filter. Then get a forth-order high-pass filter in the same way. The benefit of high order filters is clearly to be seen. Then we replace the high-pass filter with a forth-order filter. Its character was huge improved. From the bode diagram we can roughly see that the attenuating rate is higher than 30 dB per decade. It is closer to the ideological circumstance than before.

Fig. 5. Forth-order high-pass filter
3.4.2 Low-pass Filter

In the low-pass filter design, we adopt the same design idea. The principle is similar to the high-pass filter. All frequencies that are lower than a certain value are allowed through. We select 20kΩ resistors and 0.1μF capacitors as components of the low-pass filter. So that, the theoretical value of its cutoff frequency is approximate to 80Hz.

3.4.3 Band-pass Filter
A band-pass filter is a device that passes frequencies within a certain range and rejects frequencies outside that range [4].

![Band-pass Filter Diagram](image)

**Fig. 8. Band-pass Filter**

Usually, we combine a high-pass filter with a low-pass filter to work as a band pass filter. Its high cutoff frequency is decided by low-pass filter, while low cutoff frequency is decided by high-pass filter.

![Band-pass Filter Circuit](image)

**Fig. 9. Band-pass filter**

In our project, the passband is ranging from 10mHz to 80Hz.
4. Arduino Processing

The Arduino development board is an open source kit that is used for many types of projects that include both analog and digital signal processing. Its popularity in the electronics field has provided much support for the hardware and its peripherals as well as for the software libraries. 99% of all Arduino support one may require can be found on the Arduino website, <www.arduino.cc>. The picture shows the Arduino UNO, which is the most basic development board available. It is easy to use and fast to learn.
The UNO has a series of digital pins and analog pins on board. As explained later, in Section VII, the software will require a digital input for one section of the monitoring algorithm, but an analog input of the same signal for another segment of the algorithm. What we will be able to do is split the output from the amplifier circuit, ensuring there is enough current entering the board, and have one lead enter a digital pin and the other enter an analog pin. The UNO is capable of the A-D conversion, so having the analog signal from the amplifier enter the digital pin onboard will not be a problem. We will just keep in mind that we will use separate grounds for digital and analog pins [9].

5. Software design

We accomplish our software design based on a useful android toolkit, Amarino, which provides us a lot of supports on hardware communication and software samples. So we can get some sample source code easily on Amarino website and learn them fast. It makes our works more efficient when we refer its working principle to realize what our project is going to do. Our software mainly aims to solve two problems, communication with Arduino board, including receiving data from the board and process the data and display them on the phone.
5.1 Communication via Bluetooth

First of all, we should bridge our mobile with Arduino aboard if we want to show the signal on mobile device. We choose Bluetooth technology as a way to transmit signal data to android smart phone. It is no secret that Bluetooth module is an essential part of smart phone nowadays. And for our purpose of transmitting heart-beat signal to smart phone, Bluetooth is a convenient and reliable solution. For our mobile receiver, Android smart phone has a great market share in smart phone field. And its open and pervasive nature makes application development easy and efficient.

5.1.1.1 Bluetooth module setting
In order to make sure our Bluetooth module parameter match the code initialized on Arduino aboard, we have to set Bluetooth module baud rate. After connecting Bluetooth module with Arduino aboard, we can see the red LED light blinking on the module. Then we can add our Bluetooth module to our laptop by pairing Bluetooth device and installing right device drive on our computer. If we success connect Bluetooth module with our computer, we can see the name of the Bluetooth module and check it communication port.

![Bluetooth property screen](image)

**Fig. 13. Bluetooth property screen**

As we can choose the assigned communication port of Bluetooth module on Arduino IDE serial monitor, and set baud rate of Bluetooth module with this tool. Then we should look up guideline of Bluetooth module and type basic commands to set baud rate. For our Bluetooth module, we type $3$ to revoke CMD interface, E or D to check parameter of the module and type
SU,96 to set baud rate to 9600 baud.

![Arduino Serial Monitor](image)

**Fig. 14. Arduino Serial Monitor**

### 5.1.2 Amarino application

We use an open source Android application, Amarino 2.0, from Amarino website to establish a software interface to connect Arduino board. It is a new plug-in application to allow developer control Arduino aboard via Bluetooth technology. It has already provided some useful plug-in service for most sensors, such as reading data from sensor, and if some developers want to create their own plug-in to control the aboard, they can add their own event to the lists. Surely it provides a pre-defined library, MeetArdroid, to assist developer if they want to modify or add some unique events in the application. In our project, we have another application to display the data, so we just use this application to communicate with our Arduino aboard.

Now this application has even published on Google play store, the official app store supported by Google. So our potential user can get and
install this application easily if they have a smart phone based on Android 2.x OS.

![Amarino 2.0 running interface](image)

Fig. 15. Amarino 2.0 running interface

From the figure above we can see, when we run the amarino application, it will discover the entire Bluetooth devices around the phone. We just need to choose right Bluetooth module to establish connection with it. After that we can check the green LED light on Bluetooth module to make sure the connection successful. If the connection is failed, we can use Monitor to show each detail on the procedure of connection and fix the problem. When we have connected with Arduino board successfully, this application can be always in background service so we can mainly use our signal-displayed application to show the signal.
5.2 Signal Process Part

Our main purpose is to draw the heart-beat signal on our phone and we hope use this graph to provide doctor some medical reference for hear disease patient. Firstly we can get an open source application from Amarino, SensorGraph, which can read real time data transmitted from Arduino and draw the path on the phone. Generally, it is like a reading sensor data application.

The development of this application is mostly supported by two open source libraries, Amarino and Graphview. Both libraries provide powerful API to developers to revoke some useful functions directly from their packages.

From above introduce, we have learnt some information about Amarino library. It mainly provides support in communicating and controlling to Arduino. In our project, it provides APIs for connecting and receiving data from Arduino.
For the method `onStart()`, it initializes an `ArduinoReceiver` class, `arduinoReceiver`, which is responsible for catching broadcasted Amarino events, and then register a `BroadcastReceiver` to be run in the main activity thread supported by Android API. Finally, `Amarino.connect` is to establish a connection to the Bluetooth device with the given address. For the method in `onStop()`, it mainly disconnect with Bluetooth device and unregistered the receiver.

For another library, GraphView provides multiple features for developer to draw programmatically nice-looking diagrams on Android phone. It is easy to understand, to integrate and to customize it. With this library, we can control many properties of the diagrams, such as update speed, scrolling, scale, and custom paint features.

However, we find if we use the original application to display the signal, the
graph is too dense to reflect its property of heart-beat signal. And also the graph is over-centralized on the middle view of the phone. Lastly, if we can change the color and width of the stroke, we can get clearer graph, which may have a lot help for user to observe these heart-beat signal correctly.

```java
private void init() {
    mColor = Color.argb(192, 220, 20, 60);
    mPaint.setFlags(Paint.ANTI_ALIAS_FLAG);
}

public void addDataPoint(float value) {
    final Paint paint = mPaint;
    float newX = mLlastX + mSpeed;
    final float v = mYOffset + value * mScale;
    paint.setColor(mColor);
    paint.setStrokeWidth((float) 2.5);
    mCanvas.drawLine(mLastX, mLastValue, newX, v, paint);
    mLastValue = v;
    mLlastX += mSpeed;
}
```

Fig. 17. Code from Graph View

In order to get a more nice-looking diagram, we change the signal color to red (RGB 192, 220, 20, 60), time the width of stroke to 2.5(paint.setStrokeWidth (float 2.5)) and enlarge the scale of the signal to reflect larger graph (v=mYOffset + value + mScale). Finally, we change the data transmit delay from Arduino board to make the signal sparse comparing to the original one.
The project consists of several levels including signal amplification, noise filtering, signal processing, data transmission, and alarm. Since ECG signals consist of low amplitude voltages that are susceptible to a high level of noise, the signal processing part is most important that our group aim to solve. The amplification and filtering circuit enable small signal readable by Arduino board. Then, the software was built to data processing and transmission. Smartphone, which have the Bluetooth connection, is widespread in America, even all over the world. So our group chooses Bluetooth to transmit data between Arduino board and smartphone. The marriage of medical monitoring and wireless communication will make for a huge technological

6. Conclusion

Fig. 18. Comparison signal graph on modified and prototype SensorGraph
growth in the 21st century!

**Reference:**


[9] http://jjoe64.github.io/GraphView/javadoc/

Appendix

SensorGraph source code from Amarino website Copyright (c) 2010 Bonifaz Kaufmann.

package edu.mit.media.hlt.sensorgraph;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.widget.TextView;
import at.abraxas.amarino.Amarino;
import at.abraxas.amarino.AmarinoIntent;

/**
 * <h3>Application that receives sensor readings from Arduino displaying it graphically.</h3>
 * This example demonstrates how to catch data sent from Arduino forwarded by Amarino 2.0.
 * SensorGraph registers a BroadcastReceiver to catch Intents with action string: <b>AmarinoIntent.ACTION_RECEIVED</b>
 * @author Bonifaz Kaufmann - April 2010
 */
public class SensorGraph extends Activity {

    private static final String TAG = "Sensor—Graph";

    // change this to your Bluetooth device address
    private static final String DEVICE_ADDRESS = "00:06:66:66:24:2C";
    //00:06:66:4E:DE:37;

    private GraphView mGraph;
    private TextView mValueTV;

private ArduinoReceiver arduinoReceiver = new ArduinoReceiver();

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // get handles to Views defined in our layout file
    mGraph = (GraphView) findViewById(R.id.graph);
    mValueTV = (TextView) findViewById(R.id.value);

    mGraph.setMaxValue(1024);
}

@Override
protected void onStart() {
    super.onStart();
    // in order to receive broadcasted intents we need to register our receiver
    registerReceiver(arduinoReceiver, new IntentFilter(AmarinoIntent.ACTION_RECEIVED));

    // this is how you tell Amarino to connect to a specific BT device from within your own code
    Amarino.connect(this, DEVICE_ADDRESS);
}

@Override
protected void onStop() {
    super.onStop();

    // if you connect in onStart() you must not forget to disconnect when your app is closed
    Amarino.disconnect(this, DEVICE_ADDRESS);

    // do never forget to unregister a registered receiver
    unregisterReceiver(arduinoReceiver);
}
/**
 * ArduinoReceiver is responsible for catching broadcasted Amarino
 * events.
 * It extracts data from the intent and updates the graph accordingly.
 */
public class ArduinoReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        String data = null;

        // the device address from which the data was sent, we don't need it here
        // but to demonstrate how you retrieve it
        final String address = intent.getStringExtra(AmarinoIntent.EXTRA_DEVICE_ADDRESS);

        // the type of data which is added to the intent
        final int dataType = intent.getIntExtra(AmarinoIntent.EXTRA_DATA_TYPE, -1);

        // we only expect String data though, but it is better to check if really
        // string was sent
        // later Amarino will support different data types, so far data comes
        // always as string and
        // you have to parse the data to the type you have sent from Arduino, like
        // it is shown below
        if (dataType == AmarinoIntent.STRING_EXTRA) {
            data = intent.getStringExtra(AmarinoIntent.EXTRA_DATA);

            if (data != null) {
                mValueTV.setText(data);
                try {
                    // since we know that our string value is an int number we
                    // can parse it to an integer
                    final int sensorReading = Integer.parseInt(data);
                    mGraph.addDataPoint(sensorReading);
                } catch (NumberFormatException e) { /* oh data was not an integer */ }
            }
        }
    }
}
package edu.mit.media.hlt.sensorgraph;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.view.View;

public class GraphView extends View {

    private Bitmap mBitmap;
    private Paint mPaint = new Paint();
    private Canvas mCanvas = new Canvas();

    private float mSpeed = 1.0f;
    private float mLastX;
    private float mScale;
    private float mLLastValue;
    private float mYOffset;
    private int mColor;
    private float mWidth;
    private float maxValue = 1024f;

    public GraphView(Context context) {
        super(context);
        init();
    }

    public GraphView(Context context, AttributeSet attrs) {
        super(context, attrs);
        init();
    }

    private void init() {
mColor = Color.argb(192, 220, 20, 60);
mPaint.setFlags(Paint.ANTI_ALIAS_FLAG);
}

public void addDataPoint(float value) {
    final Paint paint = mPaint;
    float newX = mLastX + mSpeed;
    final float v = mYOffset + value * mScale;

    paint.setColor(mColor);
paint.setStrokeWidth((float) 2.5);
mCanvas.drawLine(mLastX, mLastValue, newX, v, paint);
mLastValue = v;
mLastX += mSpeed;

    invalidate();
}

public void setMaxValue(int max) {
    maxValue = max;
    mScale = - (mYOffset * (1.0f / maxValue));
}

public void setSpeed(float speed) {
    mSpeed = speed;
}

@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
    mBitmap = Bitmap.createBitmap(w, h, Bitmap.Config.RGB_565);
mCanvas.setBitmap(mBitmap);
mCanvas.drawColor(0xFFFFFFFF);
mYOffset = h;
mScale = - (mYOffset * (1.0f / maxValue));
mWidth = w;
mLastX = mWidth;
super.onSizeChanged(w, h, oldw, oldh);
}

@Override
protected void onDraw(Canvas canvas) {

synchronized (this) {
    if (mBitmap != null) {
        if (mLastX >= mWidth) {
            mLastX = 0;
            final Canvas cavas = mCanvas;
            cavas.drawColor(0xFFFFFFFF);
            mPaint.setColor(0xFF777777);
            cavas.drawLine(0, mYOffset, mWidth, mYOffset, mPaint);
        }
        canvas.drawBitmap(mBitmap, 0, 0, null);
    }
}
}