Wireless Diagnostics Android Application

Minqi Zhang
Hao (Neil) Li

Project Sponsor:
Charlie Brinson – eatART

Project Number: 1319
Applied Science 459
Engineering Physics Project Laboratory
The University of British Columbia
April 8th, 2013
Executive Summary

This project sought to design and test an Android Application capable of communicating wirelessly with the Titanoboa snake to display numerical data graphically. The Titanoboa snake will broadcast data through a wireless modem located in the head of the snake and be received by the Android Application which will then be displayed. Ultimately, the application is intended to be an efficient diagnostics tool to help the Titanoboa team display important values when showcasing the snake in various locations.

Motivation behind the design of the application will be to display numerical values such as battery voltage levels, motor speed, and pressure values in a graphical and intuitive way on a smart-phone. The smart-phone and Wi-Fi modem was provided by the Titanoboa team. The code was written in Java on the Eclipse IDE with the Android SDK.

The Snake consists of four modules (major break points of the snake) and each module consists of five vertebrae. Within each module consists of its own Arduino Mega and battery. Each module communicates with the rest of the snake through a RS232 serial that enables movement in a side-winding manner. The head of the Titanoboa has an additional Ethernet shield that is connected to a Wi-Fi modem. After the Android device establishes connection with the modem’s network, the App will be able to receive and transmit data.

Pre-hand testing will be done through a simulator provided by the Titanoboa team that projects simulated data through a Wi-Fi network and then received by the smart-phone. Most of the Android coding functions and classes were found through the Android Development website by Google Inc. Final tests will be conducted with the actual snake that will project its real physical data onto the application.
Table of Contents

Executive Summary ................................................................................................................... 3
List of Figures ........................................................................................................................... 5
Introduction ............................................................................................................................. 6
  General Background ............................................................................................................. 6
  Design .................................................................................................................................. 6
  Network ............................................................................................................................... 7
Technical Project Objectives .................................................................................................... 8
Scope and Limitations .............................................................................................................. 8
Discussion ............................................................................................................................... 9
  Methods ................................................................................................................................. 9
  Algorithms ............................................................................................................................ 11
Results ..................................................................................................................................... 24
Conclusion .............................................................................................................................. 25
Deliverables ............................................................................................................................. 26
Recommendations ................................................................................................................... 27
List of Figures

Figure 1: Android screen density chart ................................................................. 6
Figure 2: Titanoboa Android Application transition flowchart ................................. 7
Figure 3: Home Screen .................................................................................. 9
Figure 4: Full Snake Screen ........................................................................... 9
Figure 5: Module Screen .................................................................................. 10
Figure 6: Android XML graphical layout ............................................................. 11
Figure 7: Home Screen Activity class ................................................................. 13
Introduction

General Background
The Android platform is an open source operating system built for mobile phones and tablets that is used by millions of devices across the world. A majority of the information needed to write the Application can be found on the Android Development website. However, given that the application development on smart-phones is still a relatively new industry, taking an ideal application and putting it onto a phone was still challenging. This report will describe in detail the background design and methods used to develop an Android Application to communicate with a Wi-Fi modem for the Titanoboa.

The Titanoboa team wanted a very intuitive application that when a smart-phone with this application is pulled out of an individual’s pocket, you could easily tell how the snake was behaving. They wanted pressure values from hydraulics and voltage values from batteries to be displayed graphically. Instead of a phone screen filled with numbers, they wanted graphs and pictures that made the numerical values understandable.

Design
Our targeted smart-phone of use is the Nexus S which our sponsor has provided. It runs on Android 2.3 Gingerbread and has an 800x480 pixel display; putting this device with the high pixel density category. You can find the category for Android screen densities of various devices in the image below:

![Android screen density chart](image)

Our plan to design the application is to divide the task into three sections, a Home Screen, a Full Snake Screen, and a Module Screen. Each section would have a different perspective on how the data would be presented. The Home Screen would act as a starting point of the app. It navigates the user into the diagnostics component.

The Full Snake Screen would display the entirety of the snake, including the snake’s horizontal angle values and battery voltages for each module. Each section of the angle values will be colour coded to correspond to their respective module. In addition, the Full Snake Screen will have buttons to navigate the user to individual Module Screens of the snake.

The Module Screen will have more in-depth data such as motor speed, pressure values, and vertical angle values. It will be more comprehensive compared to the Full Snake Screen. This
gives the Titanoboa team more details for debugging and analysis if required. Below is an image of how the transitions between screens is organized.

![Figure 2: Titanoboa Android Application transition flowchart](image)

From the block diagram above, you can easily tell how the user can navigate to each section of the application. To navigate back to above sections, all Android smart-phones have physical back buttons. This allows the user to go back and forth between sections.

**Network**

One of the biggest challenges our group faced was trying to get the application to respond to data being transmitted through a Wi-Fi modem. Although the Titanboa team has provided us with a numerical Android application with numbers, transferring it over to a graphical one provided many challenges. The hardest part was dissecting the code.

The networking part of the application contains three sections, packets, network, and model. The packets section get’s the raw Wi-Fi packets and data from the Arduino Mega boards located within the modules of the snake. Once the packets are received, the network section reads the packets and also determines if the data came from the right network by checking its port number. Lastly, the model section of the network sorts the data that’s from the head, the modules, and the vertebra.
**Technical Project Objectives**

The goal of our project was to design a graphical user interface on top of a numerical Android Application provided by the Titanoboa team. How we transformed these numbers on the Android platform in an intuitive way was the challenge. To do this, there were several milestones: create a good GUI that’s readable on the tiny Nexus S screen, code the program in Java with the Android Starter Development Kit (SDK), and test the application with the simulator and the real snake.

**Scope and Limitations**

The scope of this report will describe the three sections of the application mentioned above. Each section will be looked at in detail and how it was implemented, tested, and able to wirelessly communicate with the snake. In addition, the report will provide the tested results and provide recommendations for future developers on this project.
Discussion

Methods
To design the application, our team brainstormed various ways to visually represent numerical data more visually. We would design the Home Screen to be very simple and clear with customized buttons to navigate the user around the application. The customized buttons would have a unique background designed using GIMP.

As for the Full Snake Screen, the idea we came up with is to use progress bars to represent angle values of each individual vertebrae. We would use different colour schemes to coordinate the vertebrae to their respective modules. Battery values would use vertical progress bars.
Finally, for the module screen, we have progress bars to represent angle values for vertical and horizontal actuators. To switch between modules, we implemented radio buttons. This allows transition between one module and another to be very efficient.

Other alternative methods we tried to use was using seek bars instead of progress bars. The difference would be that seek bars had a thumb nail to indicate exactly where the progress was on the bar. However, this method proved inefficient as seek bars was too thick to fit 20 of them on the Full Snake Screen.

In addition, another alternative to radio buttons was to use a swipe motion to switch between modules. This provided very fast transitions between screens. However, due to limited time, this method was not explored further.
Algorithms
To implement the above screens, you would need to create the layout with an Android XML file. Within this file, there are two sections, the xml portion and the Graphical Layout portion. Within the Graphical portion, you can drag and drop certain parts of the app from the Palette onto the screen to customize how you want the screen to look. This includes progress bars and buttons.

![Palette](Image)

**Figure 6: Android XML graphical layout**

To design our Home Screen, you want to first establish a RelativeLayout from the layout palette. Within this layout, you can drag and drop widgets in positions relative to the sides of the screen. Once you have your widgets in place, you can code the widgets to function as needed.

Each widget is characterized by certain parameters; such as width, height, and ID. These parameters must be designated in-order for your app to function. Once you have your XML file completed, you can begin to create your Application. Below is a code discussion for the Home Screen to show the different parameters of each widget.
//Creates the layout for the screen
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

  //Add a background image to the screen
  <ImageView
      android:id="@+id/imageView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/backgroundhome" />

  //Diagnostics button parameters
  <Button
      android:id="@+id/diagnostics"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_centerVertical="true"
      android:layout_marginLeft="44dp"
      android:background="@drawable/diagnostics_btn"
      android:padding="10dp" />

  //Control button parameters
  <Button
      android:id="@+id/control"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="@id/diagnostics"
      android:layout_below="@id/diagnostics"
      android:layout_marginTop="50dp"
      android:background="@drawable/control_btn"
      android:padding="10dp" />

</RelativeLayout>
To start our application, we first need to create an Activity class. An activity is a single, focused thing that the user can do. It takes care of creating a window for you in which you can place your UI with `setContentView(View);` in this case, our XML file.

```java
package ca.titanoboa;

import ca.titanoboa.model.Model;

public class TitanobcaHomeScreen extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.homescreen);
    }
}
```

Figure 7: Home Screen Activity class

Once we have our activity set up, we can find our buttons through the ID parameter and enable to perform other tasks. We could set the button to create another activity, like the Full Snake Screen class. Depending on the widgets within the XML, you could make set them to perform whatever actions the developer needs.

To create different backgrounds for buttons or have different colours for the progress bars, we need an additional XML file. This file will set different parameters for the specified widget and then can be called upon on your original layout. With this additional XML file, you can change background images of buttons, create different progress colours for progress bars, and even alter orientations of the widgets. Below is a code discussion to display progress bars in the colour green. Depending on your HEX code, you can change the colour of the progress bars to whatever colour the developer wants.
<?xml version="1.0" encoding="utf-8"?>

// tells the program that this is a widget layer related
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@android:id/background">
    <shape>
      <corners android:radius="5dip" />

      // colour for the progress bars not filled
      <gradient
        android:startColor="#ff9d9e9d"
        android:centerColor="#ff5a5d5a"
        android:centerY="0.75"
        android:endColor="#ff747674"
        android:angle="270"
      />
    </shape>
  </item>

  <item android:id="@android:id/secondaryProgress">
    <clip android:clipOrientation="vertical" android:gravity="bottom">
      <shape>
        <corners android:radius="5dip" />

        // colour for the progress bars’ secondary colour
        <gradient
          android:startColor="#80ffd300"
          android:centerColor="#80ff6b00"
          android:centerY="0.75"
          android:endColor="#a0ffcb00"
          android:angle="270"
        />
      </shape>
    </clip>
  </item>

  <item android:id="@android:id/progress">
    <clip android:clipOrientation="vertical" android:gravity="bottom">
      <shape>
        <corners android:radius="5dip" />

        // colour for the progress bars’ primary colours
        <gradient
          // HEX code of determining the colour
          android:startColor="#00CDCD"
          android:endColor="#00CDCD"
          android:angle="270"
        />
      </shape>
    </clip>
  </item>
</layer-list>
As for the coding of the full snake view, you will need to call the network functions the Titanoboa team has provided. We must call these functions in order to set the progress bars to their respective values. Below is another code discussion of the Full Snake Screen.

// To include the package of the application
package ca.titanoboa;

// All the imported classes needed to create the application
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ProgressBar;
import android.widget.RadioButton;
import android.widget.TextView;
import ca.titanoboa.model.Model;
import ca.titanoboa.model.TitanoboaModel;
import ca.titanoboa.model.module.Module;
import ca.titanoboa.model.vertebra.Vertebra;
import ca.titanoboa.network.PacketReader;
import ca.titanoboa.network.TitanoboaPacketReader;
import ca.titanoboa.packet.*;

// Beginning of the Activity
public class TitanoboaControlActivity extends Activity {

    // All the objects created
    private Model titanoboaModel;
    private PacketReader titanoboaPacketReader;
    private Thread packetReaderThread;
    private boolean packetReaderThreadStarted;

    // All the objects created
    private Model titanoboaModel;
    private PacketReader titanoboaPacketReader;
    private Thread packetReaderThread;
    private boolean packetReaderThreadStarted;
private Handler uiUpdateHandler;
private Runnable uiUpdateTask;
private int selectedModule;

// All the progress bars used to display the horizontal angle values of the actuators
public ProgressBar bar0;
public ProgressBar bar1;
public ProgressBar bar2;
public ProgressBar bar3;
public ProgressBar bar4;
public ProgressBar bar5;
public ProgressBar bar6;
public ProgressBar bar7;
public ProgressBar bar8;
public ProgressBar bar9;
public ProgressBar bar10;
public ProgressBar bar11;
public ProgressBar bar12;
public ProgressBar bar13;
public ProgressBar bar14;
public ProgressBar bar15;
public ProgressBar bar16;
public ProgressBar bar17;
public ProgressBar bar18;
public ProgressBar bar19;

// Progress bars and test to display the battery values
public ProgressBar batteryBar1;
public TextView batteryText1;
public ProgressBar batteryBar2;
public TextView batteryText2;
public ProgressBar batteryBar3;
public TextView batteryText3;
public ProgressBar batteryBar4;
public TextView batteryText4;

// Some constants
public static int MAXBATTERY = 3000;
public int MINBATTERY = 22000;
public static int MAXANGLE = 1000;

private int PORT = 12345;

// how often data updates, in ms
public static final int UPDATE_DELAY = 1;

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

    // Setting the progress bars according to their ID parameters
    bar0 = (ProgressBar) findViewById(R.id.horizontal0);
    bar1 = (ProgressBar) findViewById(R.id.horizontal1);
    bar2 = (ProgressBar) findViewById(R.id.horizontal2);
    bar3 = (ProgressBar) findViewById(R.id.horizontal3);
    bar4 = (ProgressBar) findViewById(R.id.horizontal4);
    bar5 = (ProgressBar) findViewById(R.id.horizontal5);
    bar6 = (ProgressBar) findViewById(R.id.horizontal6);
    bar7 = (ProgressBar) findViewById(R.id.horizontal7);
    bar8 = (ProgressBar) findViewById(R.id.horizontal8);
    bar9 = (ProgressBar) findViewById(R.id.horizontal9);
    bar10 = (ProgressBar) findViewById(R.id.horizontal10);
    bar11 = (ProgressBar) findViewById(R.id.horizontal11);
    bar12 = (ProgressBar) findViewById(R.id.horizontal12);
    bar13 = (ProgressBar) findViewById(R.id.horizontal13);
    bar14 = (ProgressBar) findViewById(R.id.horizontal14);
    bar15 = (ProgressBar) findViewById(R.id.horizontal15);
    bar16 = (ProgressBar) findViewById(R.id.horizontal16);
    bar17 = (ProgressBar) findViewById(R.id.horizontal17);
    bar18 = (ProgressBar) findViewById(R.id.horizontal18);
    bar19 = (ProgressBar) findViewById(R.id.horizontal19);

    // Setting the progress bars and text widgets according to their ID parameters
    batteryBar1 = (ProgressBar) findViewById(R.id.battery1);
    batteryText1 = (TextView) findViewById(R.id.batterylevel1);
    batteryBar2 = (ProgressBar) findViewById(R.id.battery2);
    batteryText2 = (TextView) findViewById(R.id.batterylevel2);
batteryBar3 = (ProgressBar) findViewById(R.id.battery3);
batteryText3 = (TextView) findViewById(R.id.batterylevel3);
batteryBar4 = (ProgressBar) findViewById(R.id.battery4);
batteryText4 = (TextView) findViewById(R.id.batterylevel4);

// Setting a maximum value for each progress bar
bar0.setMax(MAXANGLE);
bar1.setMax(MAXANGLE);
bar2.setMax(MAXANGLE);
bar3.setMax(MAXANGLE);
bar4.setMax(MAXANGLE);
bar5.setMax(MAXANGLE);
bar6.setMax(MAXANGLE);
bar7.setMax(MAXANGLE);
bar8.setMax(MAXANGLE);
bar9.setMax(MAXANGLE);
bar10.setMax(MAXANGLE);
bar11.setMax(MAXANGLE);
bar12.setMax(MAXANGLE);
bar13.setMax(MAXANGLE);
bar14.setMax(MAXANGLE);
bar15.setMax(MAXANGLE);
bar16.setMax(MAXANGLE);
bar17.setMax(MAXANGLE);
bar18.setMax(MAXANGLE);
bar19.setMax(MAXANGLE);
batteryBar1.setMax(MAXBATTERY);
batteryBar2.setMax(MAXBATTERY);
batteryBar3.setMax(MAXBATTERY);
batteryBar4.setMax(MAXBATTERY);

// Creating a new titanoboa model, described in the background section above
titanoboaModel = new TitanoboaModel();

// Creating a new packet reader, described in the background section above
titanoboaPacketReader = new TitanoboaPacketReader();

packetReaderThreadStarted = false;
// A handler to update the Application so that the data can be refreshed
uiUpdateHandler = new Handler();
uiUpdateTask = new UIUpdateTask();

// Setting the port to the correct network connection
titanoboaPacketReader.setPort(PORT);

// Individual buttons to navigate to the module view
Button mod1 = (Button) findViewById(R.id.module1)
mod1.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent(v.getContext(), module1screen.class);
        startActivityForResult(intent, 0);
    }
});

Button mod2 = (Button) findViewById(R.id.module2);
mod2.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent(v.getContext(), module2screen.class);
        startActivityForResult(intent, 0);
    }
});

Button mod3 = (Button) findViewById(R.id.module3);
mod3.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent(v.getContext(), module3screen.class);
        startActivityForResult(intent, 0);
    }
});

Button mod4 = (Button) findViewById(R.id.module4);
mod4.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent(v.getContext(), module4screen.class);
        startActivityForResult(intent, 0);
    }
});
// The Handler that updates the application
private final class UIUpdateTask implements Runnable {
    public void run() {
        Map<String, Packet> packets = titanoboaPacketReader.getPackets();

        if ((packets != null) && (!packets.isEmpty())) {
            titanoboaModel.updateDataAll(packets);

            // Setting the progress of the progress bars to the horizontal angles values of the snake.
            // Module 1
            bar0.setProgress(titanoboaModel.getModules().get(0).getVertebrae().get(0).getHorizontalSensorValue());
            bar1.setProgress(titanoboaModel.getModules().get(0).getVertebrae().get(1).getHorizontalSensorValue());
            bar2.setProgress(titanoboaModel.getModules().get(0).getVertebrae().get(2).getHorizontalSensorValue());
            bar3.setProgress(titanoboaModel.getModules().get(0).getVertebrae().get(3).getHorizontalSensorValue());
            bar4.setProgress(titanoboaModel.getModules().get(0).getVertebrae().get(4).getHorizontalSensorValue());

            // Module 2
            bar5.setProgress(titanoboaModel.getModules().get(1).getVertebrae().get(0).getHorizontalSensorValue());
            bar6.setProgress(titanoboaModel.getModules().get(1).getVertebrae().get(1).getHorizontalSensorValue());
            bar7.setProgress(titanoboaModel.getModules().get(1).getVertebrae().get(2).getHorizontalSensorValue());
            bar8.setProgress(titanoboaModel.getModules().get(1).getVertebrae().get(3).getHorizontalSensorValue());
            bar9.setProgress(titanoboaModel.getModules().get(1).getVertebrae().get(4).getHorizontalSensorValue());
        }
    }
}
// Module 3
bar10.setProgress(titanoboaModel.getModules().get(2).getVertebrae().get(0).getHorizontalSensorValue());
bar11.setProgress(titanoboaModel.getModules().get(2).getVertebrae().get(1).getHorizontalSensorValue());
bar12.setProgress(titanoboaModel.getModules().get(2).getVertebrae().get(2).getHorizontalSensorValue());
bar13.setProgress(titanoboaModel.getModules().get(2).getVertebrae().get(3).getHorizontalSensorValue());
bar14.setProgress(titanoboaModel.getModules().get(2).getVertebrae().get(4).getHorizontalSensorValue());

// Module 4
bar15.setProgress(titanoboaModel.getModules().get(3).getVertebrae().get(0).getHorizontalSensorValue());
bar16.setProgress(titanoboaModel.getModules().get(3).getVertebrae().get(1).getHorizontalSensorValue());
bar17.setProgress(titanoboaModel.getModules().get(3).getVertebrae().get(2).getHorizontalSensorValue());
bar18.setProgress(titanoboaModel.getModules().get(3).getVertebrae().get(3).getHorizontalSensorValue());
bar19.setProgress(titanoboaModel.getModules().get(3).getVertebrae().get(4).getHorizontalSensorValue());

// Battery progress bars
// Setting the progress bars to the battery values of the modules as well as setting a numerical value right below the progress bars
batteryBar1.setProgress((titanoboaModel.getModules().get(0).getBatteryLevel() - MINBATTERY));
batteryText1.setText(Double.toString(titanoboaModel.getModules().get(0).getBatteryLevel() / 1000.00));

batteryBar2.setProgress((titanoboaModel.getModules().get(1).getBatteryLevel() - MINBATTERY));
batteryText2.setText(Double.toString(titanoboaModel.getModules().get(1).getBatteryLevel() / 1000.00));

batteryBar3.setProgress((titanoboaModel.getModules().get(2).getBatteryLevel() - MINBATTERY));
batteryText3.setText(Double.toString(titanoboaModel.getModules().get(2).getBatteryLevel() / 1000.00));

batteryBar4.setProgress((titanoboaModel.getModules().get(3).getBatteryLevel() - MINBATTERY));
batteryText4.setText(Double.toString(titanoboaModel.getModules().get(3).getBatteryLevel() / 1000.00));

}  // Time delay of when the handler will refresh
uiUpdateHandler.postAtTime(this, SystemClock.uptimeMillis() + UPDATE_DELAY);

}  }

// When the activity is resumed after the user navigates away (say goes into the module view and back), we want to resume the handler and start refreshing the application again.
protected void onResume() {
super.onResume();
packetReaderThread = new Thread(titanoboaPacketReader);
uiUpdateHandler.removeCallbacks(uiUpdateTask);
uiUpdateHandler.post(uiUpdateTask);
packetReaderThread.start();
uiUpdateHandler.removeCallbacks(uiUpdateTask);
uiUpdateHandler.post(uiUpdateTask);
}

// When the user leaves the activity, we want to pause the handler and stop refreshing the data.
protected void onPause() {
packetReaderThread.interrupt();
uiUpdateHandler.removeCallbacks(uiUpdateTask);
super.onPause();
}

}  // End of Full Snake Screen code
As for the module view, there are five buttons listed on bottom of Full Snake Screen which can direct to the module view section. Each button is labeled with a specified name which links to the specified module view.

The module view section is supposed to show all parameters of interest received by the application. They includes number of modules connected, battery level, motor speed, hydraulic pressure, horizontal set points and positions, horizontal high and low calibration, and vertical set points and positions, vertical high and low calibration. The above data is as a packet sent through the Titanoboa head. The application is supposed to receive the packet via wireless communication. The packet data is then analyzed and displayed on the module view section.

The basic principle of designing the module view section is to keep it clean and clear while make it an effective application. The module view interface is an essential and necessary part of the program. Considering the structure of the whole application, it is logical to put module view section as a sub-view of diagnostic section. Since the function of diagnostic part is to receive and display the data, therefore module view section is arranged under the diagnostic section. Further recommendations and improvements are discussed at last.
Results

The end result was a successful test of the application using the simulator. The progress bars fluctuated according to the simulator and the values displayed the correct projected data. Below is a YouTube link to a video of the application in use.

http://www.youtube.com/watch?v=CGEw-tWsSqY
Conclusion
The primary conclusion of this report is that we were able to create an Android Application using Eclipse that convert numerical data and making it graphical and communicating with a transmitter through Wi-Fi. We were able to complete this object through sequential steps of first displaying static data, then dynamic data, and finally the simulated data. Although we have not yet tested the application on the real Titanoboa, we are confident it will prove successful.

The most significant result of our application was being able to allow it to respond to the simulator. We’ve ran the simulator multiple times and the results proved to be successful. Each time it was run, the results were accurate and corresponded to the simulator and the numerical application. The numbers were the same when compared.
Deliverables
Below is a list of all of the initial deliverables with their current status. Because we have yet to test our application on the real Titanoboa, we were unable to further improve our program.

- Android Application and its source code
  - Due to some minor issues, we are still having trouble making the Module Screen connect with the Wi-Fi.

- Detailed report on how the application works

- Galaxy S smart-phone used for testing
  - This device was originally borrowed from the Titanoboa team in the beginning when the project was first started.

- Wi-Fi modem used for testing
  - The modem was originally borrowed from the Titanoboa team along with the Galaxy S.
Recommendations
Our recommendations are as follows, not necessarily in order of importance:

1. Within the Module Screen, replace the radio buttons with a swipe command to navigate between modules allowing more efficient transition.

2. If more modules needed to be added to the snake, you could implement its own set of radio buttons or add a swipe command onto the Full Snake Screen so the entirety of the snake can be preserved shown.

3. Transition the application for tablets. Currently, a lot of the screens are only suited for the single device Nexus S.