# Final Design Project: Bluetooth and Andriod Apps
# EE/CSE 474 Lab 5

Daisy Xu, Yunie Yi, Abigail Santos


December 9, 2016

# Table of Contents:

# 1. INTRODUCTION

The purpose of this laboratory was to use and integrate the knowledge and resources we gained throughout the previous labs into one project. This report serves to provide information on the new elements added into the project, and will not go into detail about previously used components and devices. For our project, we built an Android application that acts as a remote control for our BeagleBone tank. The app has four control buttons: Forward, Backward, Clockwise (rotates left) and CounterClockwise (rotates right). There is also a switch to turn on automatic driving, in which the tank will drive unprompted, using distance sensors to avoid running into objects. As an additional functionality, we implemented a voice control option that will allow the BeagleBone to execute the commands when certain phrases are said. The Android app was designed to run on a Samsung Galaxy S5 using the RN-42 SparkFun Bluetooth Modem to communicate with the BeagleBone.

# 2. DISCUSSION OF THE LAB & SPECIFICATIONS

## 2.1 Startup Script

In order to have more flexibility in moving the BeagleBone tank, we needed to execute a startup script to allow the tank to run untethered. The purpose of the startup script is to load all the linux terminal commands that we manually entered to execute our programs. To do this, we simply created a bash file with a sequence all of these commands (e.g. echo to initialize ports, gcc to compile code, running executables) in the file. Using this .sh file, we created a service routine such that the file would be executed upon bootup of the BeagleBone.

Because the BeagleBone constantly changes the exact folder name of an initialized PWM on bootup, the bash file also had to account for this. For example, whenever the PWM in P9-14 is created, it creates a folder called pwm_test_P9_14_XX, where “XX” are two numbers that erratically change. The same problem occurs when creating the analog pins used by the distance sensors. Using the **ls** and **grep** commands, the startup script finds the correct path for the PWM and analog pin folder locations, and passes them along to the programs that use them, namely handler.c and motor.c.

## 2.2 Bluetooth Implementation

A bluetooth modem is necessary to control the BeagleBone tank wirelessly by connecting the BeagleBone to another device. For this project, we used the RN42 modem as seen in **Figure 1**. The RX(receiver) and TX(transmitter) pins are used to communicate between the two devices, as seen in **Table 1**, implementing a UART communication protocol. These ports support serial communication when both sending and receiving. In order to configure the ports to support the same baud rate of 115200 at both ends, we used the python library Pyserial to ensure the bluetooth and beaglebone were properly connected before running our program files.


**Figure 1: Pinout of RN-42 SparkFun Bluetooth Modem**

| Bluetooth Pin | Description | Connection with Beaglebone |
|---|---|---|
| VCC | Power Supplier | VDD 3.3 V |
| GND | Ground | DGND |
| TX - 0 | Transmits data to BeagleBone | UART4-RX |
| RX - 1 | Receives data from BeagleBone | UART4-TX |

**Table 1: Bluetooth Pin Descriptions and Connections**

Our motor.c code was slightly modified to account for the bluetooth integration. It reads an int from the /dev/ttyO4 bluetooth folder on the BeagleBone, and this int corresponds to a command that tells the tank to drive a certain direction. These commands are summarized below in **Table 2**. A description of our methods in motor.c and handler.c, which controls tank movement, will not be discussed in depth but the reader may refer to previous reports for a more information.

| Command | Direction |
|---|---|
| 1 | Moving forward |
| 2 | Moving backward |
| 3 | Rotating right |
| 4 | Rotating left |

| | |
|---|---|
| 5 | Auto Drive: tank moves depending on the distance sensors by avoiding objects as necessary |

**Table 2: Key of Commands Controlling the Motor in motor.c**

## 2.3 Android Application as Remote Controls

To control the tank, we built an Android application that communicates to the tank via bluetooth. Screenshots of the app as it executes commands may be seen in **Figures 2 and 3**. At a lower level, each button writes one of the five commands as shown above in Table 2 to the same /dev/ttyO4 folder in the BeagleBone. A description of the buttons is provided below in **Table 3**. The application, named BeagleBoneControl, was written using the Android SDK in Android Studio.

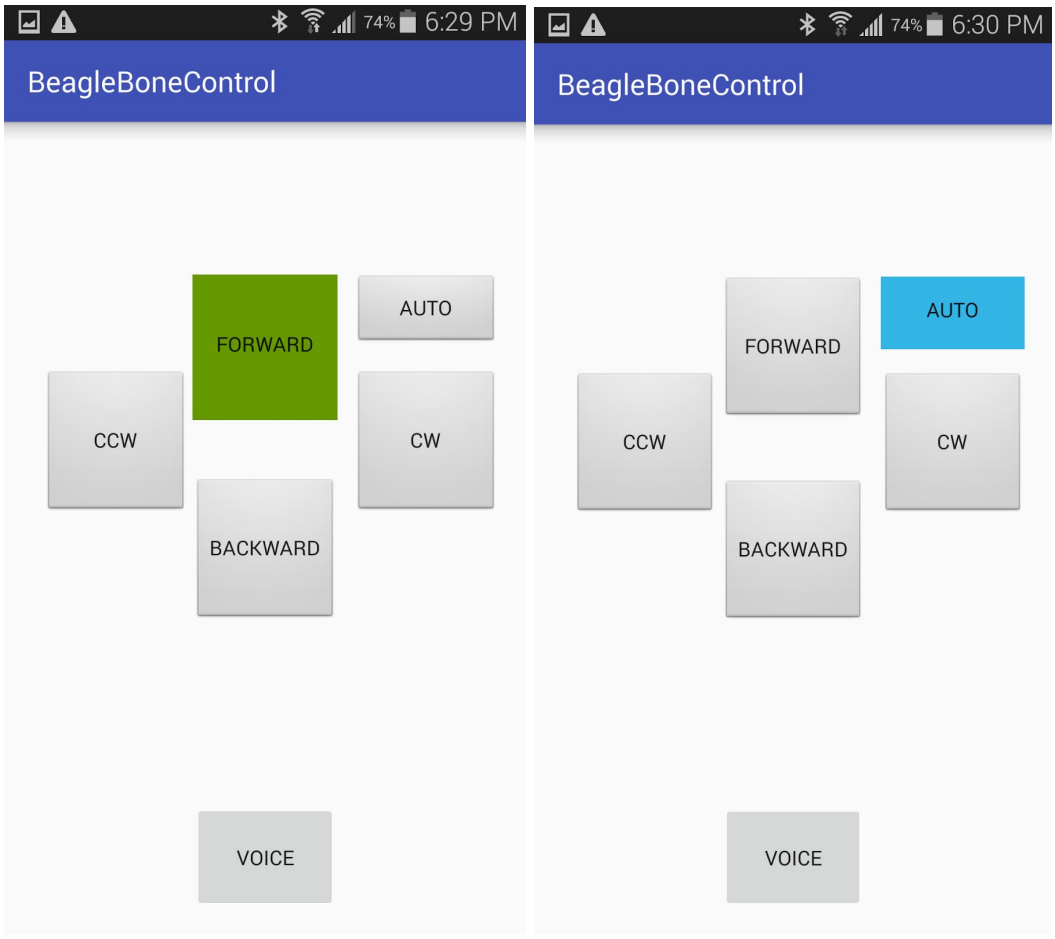


**Figure 2: Screenshots of BeagleBoneControl App with Forward button and Auto button Being Pressed**

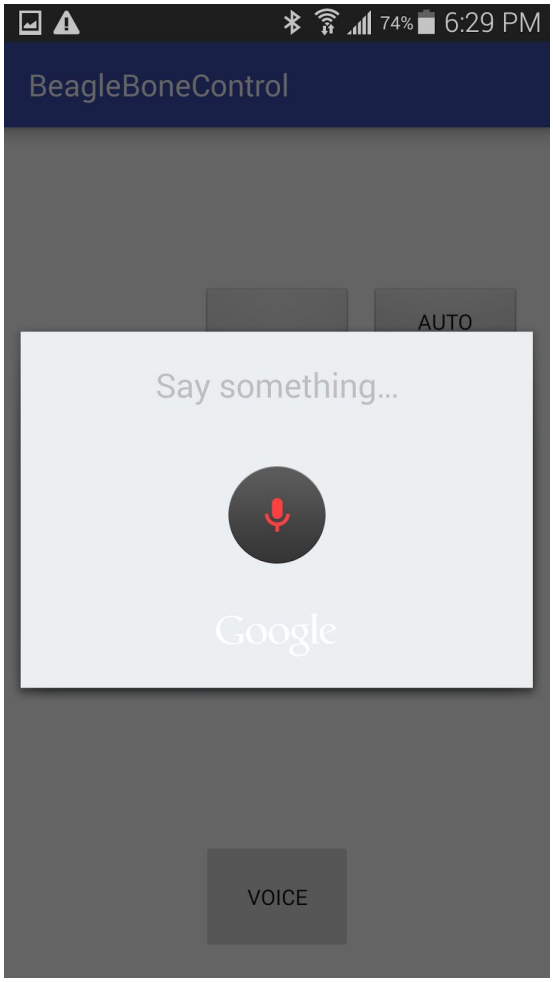


**Figure 3: Screenshots of BeagleBoneControl App Accepting a Voice Command**

| **Button** | **Description** |
|---|---|
| FORWARD | Sends Command 1 to the BeagleBone while button is pressed to move tank forward. Tank stops moving upon release. |
| BACKWARD | Sends Command 2 to the BeagleBone while button is pressed to move tank backward. Tank stops moving upon release. |
| CCW | Sends Command 3 to the BeagleBone while button is pressed to rotate tank counterclockwise (right). Tank stops moving upon release. |
| CW | Sends Command 4 to the BeagleBone while button is pressed to rotate tank clockwise (left). Tank stops moving upon release. |

| | |
|---|---|
| AUTO | Sends Command 5 to the BeagleBone once button is pressed to start automatic driving. This button is "clicked" so it must be pressed again to stop the tank. |
| VOICE | Takes a voice input that will send the corresponding issued command to the BeagleBone until otherwise told not to. If any key phrases are said, it will read and execute only the first key phrase heard. Inputs with no key phrases will cause the tank to stop. The following phrases translate to the following commands:<br><br>Forward: "Go", "Forward"<br>Backward: "Go Back", "Back", "Backward"<br>CCW: "Go Right", "Right"<br>CW: "Go Left", "Left" |

**Table 3: Summary of BeagleBoneControl App Buttons**

Though our app itself consists of several dozen files, only two, MainActivity.java and WriteTask.java, contain most of the logic and setup. These files are summarized below in **Table 4**.

| **File Name** | **Description** |
|---|---|
| WriteTask.java | Connects to the bluetooth socket and writes the command to the beaglebone. |
| MainActivity.java | Handles all the functionalities and interface designs as well as finding the bluetooth device between the beaglebone and Android phone. It sets all commands from button press and translate the action to output data from device. For the voice recognition function, it translate the phrases from user to the commands. |

**Table 4: Summary of Java Codes Used in Android Application**

# 3 Test Procedure & Cases

## 3.1 Startup Script

**Case 1: Verify Script Functionality**

To verify that our startup script contains all the necessary sequences of commands that were normally executed by hand in the linux terminal, we first ran just the script to see if it properly ran our previous motor codes in Lab 4.

**Case 2: Verify Service Routine and Script Functionality**

After confirming the script itself was working properly, we had to verify the service routine was properly connected to it. We first tested running the service routine manually, and after confirming it did, we had to test that the service routine correctly ran on startup.

## 3.2 Bluetooth Implementation

**Case 1: Transmit and Receive Data via Bluetooth Between BeagleBone and Computer**

To implement the bluetooth, as previously mentioned, we used the Python library, Pyserial, to configure the ports and read/write to them. On one end we used a computer that, after pairing with our bluetooth named CAS, continually sent an example string (e.g. "hi") to the bluetooth RX folder /dev/tty.CAS. Simultaneously, on the BeagleBone end, we continually read from the bluetooth TX folder /dev/ttyO4. The goal of this test is to confirm very basic communication via bluetooth and that baud rates were correctly configured before more complicated transmissions with our Android App.

**Case 2: Write Data Received via Bluetooth to Output Text File**

In Case 1, we had difficulties in seeing if all of the data sent was properly received, partially due to the lag in sending/receiving. To rectify this, we modified the receiving code used by the BeagleBone to also output received data into a .txt file. The .txt file can be easily checked to confirm signal integrity.

## 3.3 Android Application as Remote Controls

**Case 1: Verify Data Received is Data Sent**

Our Android app sends an ASCII character command, and our motor.c must read this ASCII character correctly. Because the bluetooth defaults to a baud rate of 9600, it must be changed to 115200 in order to properly read ASCII. Therefore, we must verify that any characters sent and received are the same, just as we did for the Bluetooth Implementation between the BeagleBone and a computer.

**Case 2: Verify Correct Commands Executed Upon Button Push**

After verifying the communication network works as intended, we needed to verify that the user interface properly worked as intended and sends and executes the commands for each button. This means the app interface, bluetooth, and motor C codes must we working in tandem with each button press.

**Case 3: Stop Executing Commands Upon Button Release**

In making our Android application, we needed to find a way to detect when the user has released a button and is no longer sending commands. Thus, we decided to use “0” as another command to be interpreted as a “stop” from the tank. This “stop” was set as a default command to be sent if no other commands are being sent, meaning that if a button is released, any directional commands currently being executed should end and the tank should come to a stop.

**Case 4**: **Voice Recognition**

Voice commands were designed such that the given command is executed until the user says otherwise and should not stop without being given another command. This means that if the user says another voice command or manual command (button press), the current command should stop and the new command should be transitioned into.

# 4. RESULTS

## 4.1 Startup Script

The startup script and service routine was properly executed upon bootup of the BeagleBone, running as intended. Each file was automatically compiled and run without error.

## 4.2 Bluetooth Implementation

The communication between the Beaglebone UART ports and bluetooth device was successful. Data sent to the bluetooth was successfully read on the BeagleBone end with no issue.

## 4.3 Android Application as Remote Controls

Each button in our Android App correctly sent its corresponding command and voice control was accurate in detecting valid commands. Sent commands were received by the BeagleBone and properly executed by our C codes, and as a result the tank navigated exactly as intended using buttons, auto, and voice commands.

# 5. ERROR ANALYSIS

Besides common errors due to syntax, unplugged wires, and dead batteries, there were several mishaps throughout the development process of this project. First, when creating the startup file, we noticed our executables were not being run. After much debugging, we discovered that the “make” command was not valid in a startup script, as it only compiled the first code in the Makefile and no others. After replacing the “make” line with gcc commands to compile each used code individually, we successfully got the startup script and service routine to run correctly.

Another error occured when trying to establish basic communication with a reader and writer via bluetooth. Our code seemed to be correct, but for some reason on the receiver end, data was never received. Then we discovered that the wiring for our bluetooth modem was incorrect - we had plugged the TX pin into the BeagleBone TX, and the RX pin into the BeagleBone RX, assuming that TX meant it was receiving and was supposed to go into the transmitting pin, and the same thinking for RX. In actuality, TX contained transmitting data and belongs in the receiving BeagleBone pin RX. After switching these two wires, communication worked perfectly.

We also had trouble with the distance sensors reading highly fluctuating values. We determined that this was because the analog pins in the BeagleBone were getting overloaded by the output voltage from the sensors. By adding in potentiometers in series with each sensor, we were able to achieve much lower and more stable readings from each sensor, which made auto-driving run much smoother.

Our last major error occured from the configuration of the bluetooth ports, namely the baud rate. While during basic tests it ran correctly, when we tried to run our motor.c code as a receiver from our Android App, we noticed it was reading garbage. This is because it was reading HEX instead of ASCII because we no longer had the baud rate set. After properly setting it to 115200, the default on Android, communication was once again successful.

## 6. Summary and Conclusion

In this lab, we successfully integrated both hardware and software to create an Android app that functions as a remote control for our tank. Completing this lab meant building a knowledge base of how to use single-board computers, sensors and motors, Bluetooth and UART communication protocol, timing and interrupts, and the Android SDK, while getting highly refined debugging skills along the way.

We encountered numberless bugs during development, from unstable distance sensors to broken motors, but we were able to overcome these challenges and create a fully functioning system. The Android App accurately worked as a controller for our tank using a button interface, while also supporting an autodrive feature and voice commands. Future improvements however, can be made to the system. Older parts should be replaced, and the user interface of our app could also be made more appealing. If possible, the lag between the phone and the tank should also be reduced. Overall, after completion of this lab and this quarter, the team stands much more comfortable with embedded systems and the endless possibilities one can achieve given the right resources.