Colorado School of Mines

Android Controlled Robot

To: John Steel

From: Michael Dephillips, Robert Malone

Team #: 414

Date: 12/08/2011

Re: Lab #: 7, Student Design Project, Android Controlled Robot

Problem Statement:

The objective for this lab was to wirelessly control our robot . The Android Smartphone communicates to the robot through Bluetooth. The first steps were researching if the project was feasible in the first place. Then we researched a suitable Bluetooth module and it will interface with our robot.

Approach:

a) Hardware

After research, the module we decided upon was the Bluetooth Mate Gold from Sparkfun shown in figure 2.0, mostly due its known history of being able to work with the Arduino. The Bluetooth Mate Gold has a transmitting distance of 106 meters in open air, and is directly compatible with the Arduino Pro, Pro Mini, or LilyPad Mainboard. In future applications, we would not need a Bluetooth module with such a long range; however, our instructor, John Steel, encouraged us to get the long-range module. The Bluetooth Mate Gold provides connections for Request to Send(RTS), Clear to Send(CTS), VDC, GND, RX(Receive), and TX(Transmit). The Request to Send goes high (+3.3V) when the Bluetooth module wants to send data, and the data is not sent until the Clear to Send input is high (+3.3V). The Bluetooth Mate Gold also has on-board voltage regulators, so it can be powered from a 3.3 to 6VDC power supply, which in this case was supplied from the Arduinos 3.3V power output. The RX and TX connections had to be swapped when connected to the Arduino for some reason, possibly due to a manufacturing error.

As we researched serial communication on the HandyBoard, we could not find any documentation about how to interface a Bluetooth module. The only way that the Bluetooth module could be connected to the Handyboard was through the serial port, and no information was available to us on the API that supported serial communication in Interactive C. The solution we came up with was to use an Arduino microcontroller as a host controller to the Handybaord.

We used the Arduino Duemilanove, which contains the necessary RX and TX pins in order to communicate with the Bluetooth module. The way process works is by sending the commands from the Android phone to the Bluetooth module, the Arduino would receive the commands, and then send a 3 pin signal to the Handyboard's input pins, in turn telling

the robot to drive, turn, stop, turn left, turn right, or speak. This approach involved using three different code platforms in order to execute the project. We decided to short the CTS, and RTS pins in order for the Arduino to constantly be receiving available serial data.

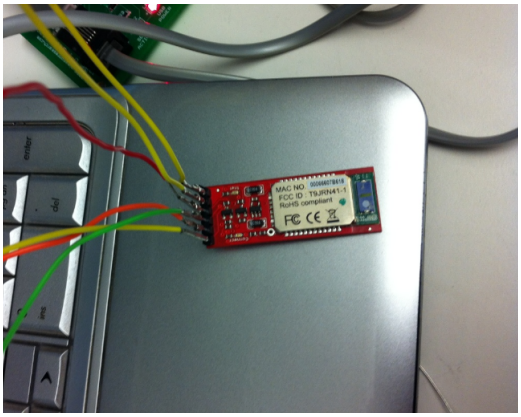


Figure 2.0 Picture of the Bluetooth Mate Gold.

In order to connect the Arduino to the Handybaord, three of the digital output pins of the Arduino were connected to the digital input pins of the Handyboard along with a shared ground between the two. The Arduino was initially powered by the USB connection for a power supply of 5VDC, but later was supplied by the 5VDC from the Handyboard in order to operate wirelessly. It was also later found that if the USB was connected to the Arduino that it wouldn't allow the Bluetooth Mate Gold to receive communications from the Andriod.

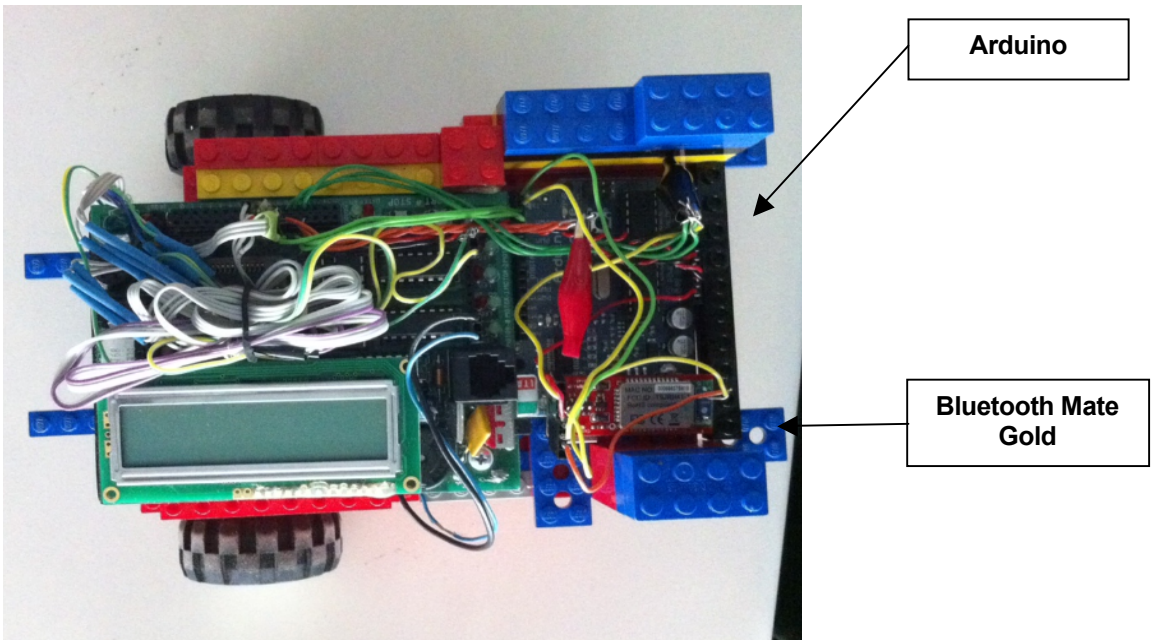


Figure 1.0 Picture displaying the robot configuration.

b) Algorithm

The algorithm consists of three parts. The Android app, the Arduino , and the Interactive C code. They communicate to each other in that order.

The Android app uses an API called MeetAndroid. It is an open source library that wraps the Bluetooth Android API into easily called functions like connect(device_id) and send(flag, byte_data). It can be downloaded at http://www.amarino-toolkit.net/index.php/download.html. The app's GUI is displayed below in figure 3.0.

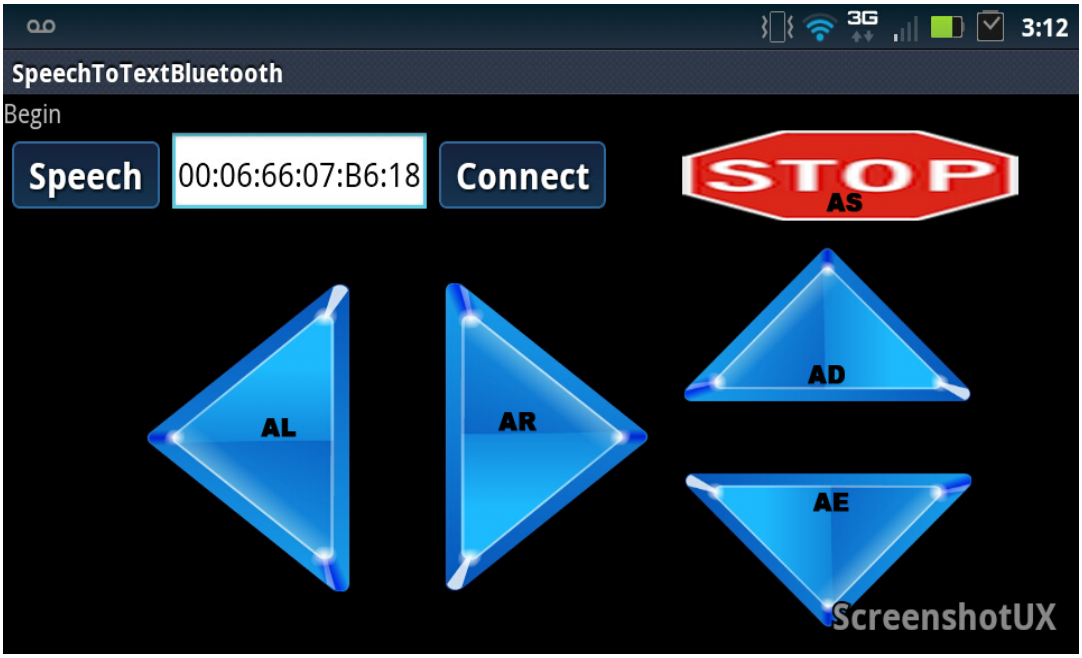


Figure 3.0 Screen shot of the Andriod GUI used to control the robot.

Before you can control the robot, you must click the connect button, with the correct MAC address in the text box. The MAC address can be found on the Bluetooth module. The program will then ask the user for a pin number, and you must enter "1234". The texts on the buttons were added for clarity. The text is the serial command that was sent to the Bluetooth Module. The data package for a left button click looks like this "AL[CheckSum]". A checksum is a value that the receiver uses to check if the data was sent correctly. The Speech Recognition is enabled when you press "Speech" button. The program then checks if the spoken word is the same as one of our command options [drive, left, right, stop, speak, reverse]. If it is, then the app sends a serial command in the form of "A[char_command][CheckSum]".

The command then travels through the airwaves, is received by the Bluetooth module, and sent serially to the Arduino board.

The Arduino begins by setting up the serial port to communicate at an 115,200 baud rate. This value was chosen, because both the Bluetooth module and the Arduino can communicate at this rate. The main loop of the Arduino waits until it has data in the serial port available for reading. If the first read byte is an 'A' flag, then it is a command sent from the phone. The next byte determines the type of command. Depending on the type

of command, the Arduino will then set three digital ports to a value 001-110 (binary) for 100ms, so the HandyBoard has enough time to recognize the command. 100ms was chosen, because it allows the HandyBoard to read the command once and only once.

This system may seem very redundant to the reader. That is, because it is. We originally built our vehicle robot with the HandyBoard, but the HandyBoard does not have sufficient documentation on its serial communication. Therefore, within the time and scope we were given, it was much easier to connect the Bluetooth module to the Ardruino board, and then use digital i/o pins to communicate to the HandyBoard. There is enough documentation on digital i/o of the HandyBoard to accomplish this task. Ideally, and for future projects, we would only need one microcontroller board.

Next, the HandyBoard IC code is explained. The HandyBoard main loop checks the state of three digital input pins. If these digital input pins reach a state besides all low (0V), then the robot should do a function. The table below shows the corresponding functions and their values:

Robot Command	Digital Input State (pin 10,11,12)
Drive	110
Turn Left	101
Turn Right	100
Stop	011
Speak	010
Reverse	001

Table 1.0 Table showing the corresponding functions and their values received at the digital input pins of the Handyboard.

The functions will not be gone into detail, because they are similar to what we have done in previous labs. Code for all three programs are in the Appendix.

Results:

a) Speech text versus push button GUI control

Since there wasn't a maze to run the robot through in this lab, there isn't a lot to document as far as results are concerned. The speech to text was a little slow since an Internet connection had to be used in order for it to work. The test times found in the lab are displayed in table 2.0 below. The time was dependent on the Internet connection being used, so if there were a strong connection being used for this feature it would work almost instantaneously. The push button GUI was instantaneous however, only taking the amount of time for the Android to transmit the Bluetooth signal. Occasionally, we also experienced some, what seemed random, connection failures between the Android phone and Bluetooth module. The Bluetooth module green connection light would shut off every once in a while, in which the Android would have to reconnect. The reasons for the connection failures were not discovered, but could have possibly been due to interference or battery power.

Speech to text times from the lab in seconds
5.5
4.5
4.3
3.9
4.8

Table 2.0 Times found in the lab for the robot to respond to speech to text function.

Conclusions:

a) How well did our robot perform?

Our robot performs very well. The response from the Android app through the Bluetooth to the HandyBoard is very quick, instantaneous to the human eye. The robot has pretty good dexterity, and can make it through the maze in lab in less than 30 seconds. This is when the phone's buttons controls it.

b) What was the biggest surprise about this project?

We believe the biggest surprise was the speed of Bluetooth transmission. It is almost instantaneous. The benefit of this opens up a whole world of implications for controlling robots wirelessly in real-time.

c) What did we do well?

We did the wiring and hardware setup very well. We researched the Bluetooth module, the Arduino board, and the HandyBoard very meticulously so that we were able to deliver a solution with limited wires.

d) What did we not do well?

With time constraints, we did not get the robot turning as accurate and realistic as possible. We also were not able to achieve real-time speech recognition to control the robot.

e) What would we do different?

We would make an iPhone app as well. The iPhone has more third party APIs that would allow us to do speech recognition in real time. Unfortunately, the Android phone uses Google through an HTTP connection to do speech recognition, so there is a lag between speech and recognition. This fact limits the implications of the project, but there are ways to do speech recognition in real-time in both platforms. This is something we would do different if we had more time. We would also not use the HandyBoard at all. There is really no need for the HandyBoard except that our vehicle robot already implemented it.

f) Implications

There is a vast range of implications with this project. Here are a few ideas that we came up with:

1) Camera mounted on robot sending images back to the phone.
2) IR Sensor sending object data to the phone. Build virtual image of a maze.

3) Draw a path on the Android phone and then the robot follows that path
4) Program your microcontroller through your Android phone or computer
5) Speech Recognition on Android to display message on LCD screen

With these first steps we made with interfacing the robot to the Bluetooth all of these other projects now seem feasible.

Appendices:

Appendix A – Android Activity Code

```
package com.mines.robotics;

import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.content.pm.ResolveInfo;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.speech.RecognizerIntent;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import at.abraxas.amarino.Amarino;

public class SpeechToTextBluetoothActivity extends Activity {

        private static final int VOICE_RECOGNITION_REQUEST_CODE = 1523;
        private String m_DEVICE_ADDRESS = "";
        private boolean m_Connected = false;

        // signals to robots and their human readable equivalent
        private static final char DRIVE = 'D';
        private static final char LEFT = 'L';
        private static final char RIGHT = 'R';
        private static final char STOP = 'S';
        private static final char SPEAK = 'T';
        private static final char REVERSE = 'E';

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

    // Get display items for later interaction
    Button speakButton = (Button) findViewById(R.id.speech_btn);

    // Check to see if a recognition activity is present
    PackageManager pm = getPackageManager();
```

```
        List<ResolveInfo> activities = pm.queryIntentActivities(
                new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH), 0);
        if (activities.size() != 0) {

        } else {
           speakButton.setEnabled(false);
           speakButton.setText("Recognizer not present");
        }

        // load last device setting
             SharedPreferences prefs =
PreferenceManager.getDefaultSharedPreferences(this);
             EditText deviceId = (EditText)this.findViewById(R.id.device_id);
             deviceId.setText(prefs.getString("DEVICE_ID", "Enter Device Id"));
    }

    /**
     * Handle the results from the recognition activity.
     */
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
       if (requestCode == VOICE_RECOGNITION_REQUEST_CODE && resultCode ==
RESULT_OK) {
             String strOutput = "";
           TextView output = (TextView)this.findViewById(R.id.txt_output);
           // Fill the list view with the strings the recognizer thought it could have heard
           ArrayList<String> matches = data.getStringArrayListExtra(
                RecognizerIntent.EXTRA_RESULTS);
           for( int i = 0; i < matches.size(); i++ ) {
             strOutput = matches.get(i).toString();
           }
           String[] commands = {"drive","left","right","stop","speak","reverse"};
           char[] commandCodes = {'D','L','R','S','T','E'};
           String wordChoice = strOutput;
           int wordToSend = -1;

           // loop through google's word choices and see if any of them match a command
           for(int i = 0; i < matches.size(); i++) {
             String match = matches.get(i);
             for(int j = 0; j < commands[i].length(); j++) {
                        // if we find a match, select the command character and exit loop
                        if( match.equals(commands[j])) {
                                   wordChoice = commands[j];
                                   wordToSend = j;
                        }
             }

           }

           // old algorithm
```

```
    //      float maxProb = 0f;

        /* loop through each command word and determine which word most
resembles
        // the intended command
        for(int i = 0; i < commands.length; i++) {
          int counter = 0;
          // sum all letters in common
          for(int j = 0; j < commands[i].length(); j++) {
                  if( commands[i].contains(strOutput.substring(j,j+1)) ) {
                          counter++;
                          // place more weight on the first letter of the word
                          if( j == 0 ) {
                                  counter = counter + 3;
                          }
                  }
          }
          // find probability that the word matches
          float prob = (float)counter/(float)commands[i].length();
          wordChoice = wordChoice + " " + prob;
          // update the most probable word
          if( prob > maxProb ) {
                  wordChoice = wordChoice + " " + commands[i];
                  maxProb = prob;
                  wordToSend = i;
          }
        }

        // hard coded exception
        if(strOutput.equals("write")) {
          wordChoice = wordChoice + " " + "right";
          wordToSend = 2;
        }*/

        if( wordToSend != -1 ) {
                // output results
                  output.setText(wordChoice);

                          // I have chosen random small letters for the flag 'c' for
sending a command to the robot
                          // you could select any small letter you want
                          // however be sure to match the character you register a
function for your in Arduino sketch
                          if(m_Connected) {
                                  Amarino.sendDataToArduino(this,
m_DEVICE_ADDRESS, 'A', commandCodes[wordToSend]);
                          }
        }
        else {
          // output failed results
```

```
                output.setText("No matching command");
            }
        }

        super.onActivityResult(requestCode, resultCode, data);
    }

    /**
     * Return to login screen
     */
    public void speechConvertButton(View v) {
        // call speech to text activity
            startVoiceRecognitionActivity();
    }

    /**
     * Send signal to drive robot
     */
    public void driveButton(View v) {
            if(m_Connected) {
                    Amarino.sendDataToArduino(this, m_DEVICE_ADDRESS, 'A', DRIVE);
            }
    }

    /**
     * Send signal to have robot turn left
     */
    public void leftButton(View v) {
            if(m_Connected) {
                    Amarino.sendDataToArduino(this, m_DEVICE_ADDRESS, 'A', LEFT);
            }
    }

    /**
     * Send signal to have robot turn right
     */
    public void rightButton(View v) {
            if(m_Connected) {
                    Amarino.sendDataToArduino(this, m_DEVICE_ADDRESS, 'A', RIGHT);
            }
    }

    /**
     * Send signal to stop the robot
     */
    public void stopButton(View v) {
            if(m_Connected) {
                    Amarino.sendDataToArduino(this, m_DEVICE_ADDRESS, 'A', STOP);
            }
    }
```

```
	/**
	 * Send signal to reverse the robot
	 */
	public void reverseButton(View v) {
		if(m_Connected) {
			Amarino.sendDataToArduino(this, m_DEVICE_ADDRESS, 'A',
REVERSE);
		}
	}

	/**
	 * Connect bluetooth device id in edit text field
	 */
	public void connectButton(View v) {
		EditText deviceId = (EditText)this.findViewById(R.id.device_id);

		// save address for later use
		m_DEVICE_ADDRESS = deviceId.getText().toString();

		// connect to bluetooth device
		Amarino.connect(this, m_DEVICE_ADDRESS);

			// save state
			PreferenceManager.getDefaultSharedPreferences(this)
				.edit()
					.putString("DEVICE_ID", m_DEVICE_ADDRESS)
				.commit();

			// let app know that it has been connected
			m_Connected = true;
	}


	/*
	 * Stops bluetooth connection
	 * (non-Javadoc)
	 * @see android.app.Activity#onStop()
	 */
		@Override
		protected void onStop() {
			super.onStop();

			// if it is connected
			if( m_Connected == true ) {
				// stop Amarino's background service, we don't need it any
more
				Amarino.disconnect(this, m_DEVICE_ADDRESS);
			}
		}
```

```
    /**
     * Fire an intent to start the speech recognition activity.
     */
    private void startVoiceRecognitionActivity() {
        Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);

        // Specify the calling package to identify your application
        intent.putExtra(RecognizerIntent.EXTRA_CALLING_PACKAGE,
getClass().getPackage().getName());

        // Display an hint to the user about what he should say.
        intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "Robot Instruction");

        // Given an hint to the recognizer about what the user is going to say
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
RecognizerIntent.LANGUAGE_MODEL_WEB_SEARCH);

        // Specify how many results you want to receive. The results will be sorted
        // where the first result is the one with higher confidence.
        intent.putExtra(RecognizerIntent.EXTRA_MAX_RESULTS, 5);

        // Specify the recognition language. This parameter has to be specified only if the
        // recognition has to be done in a specific language and not the default one (i.e.,
the
        /* system locale). Most of the applications do not have to set this parameter.
        if (!mSupportedLanguageView.getSelectedItem().toString().equals("Default")) {
            intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE,
                    mSupportedLanguageView.getSelectedItem().toString());
        }*/

        startActivityForResult(intent, VOICE_RECOGNITION_REQUEST_CODE);
    }

}
```

Appendix B – Ardruino Duemilanove ATmega328 Code

```
/*
  Receives robot commands from your phone
  After it gets a message the led will blink
  and a digital signal will be sent to the
  HandyBoard for 100ms.
 */
```

```
int onboardLed = 13;
int counter = 0;
int incomingByte;
/*
 * This method is called constantly.
 * note: flag is in this case 'A' and numOfValues is 0 (since test event doesn't send
any data)
 */

// digital output pins to represent bit0, bit1, and bit2 of input to Handy Board
// These output pins go to pins 10, 11, and 12 on handy board
int bit0 = 2;
int bit1 = 3;
int bit2 = 4;

// blink LED (labeled L) for a certain time
void flushLed(int time)
{
 digitalWrite(onboardLed, HIGH);
 delay(time);
 digitalWrite(onboardLed, LOW);
 delay(time);
}

void setup()
{
 // use the baud rate your bluetooth module is configured to
 // ours we programmed to work in 115,200 baud, because the
 // ardruino only works with 57k or 115k bauds
 Serial.begin(115200);

 // on board led
 pinMode(onboardLed, OUTPUT);

  // set all color leds as output pins
 pinMode(bit0, OUTPUT);
 pinMode(bit1, OUTPUT);
 pinMode(bit2, OUTPUT);

 // just set all leds to high so that we see they are working well
 digitalWrite(bit0, HIGH);
 digitalWrite(bit1, HIGH);
 digitalWrite(bit2, HIGH);

 // start up program LED flash
 flushLed(1000);
}

void loop()
{
```

```
 // see if there's incoming serial data:
if (Serial.available() > 0) {

 // read the oldest byte in the serial buffer:
 incomingByte = Serial.read();

 // if it's a capital A, we want to do the Action Command
 if (incomingByte == 'A') {
  // Quickly flash the LED to know we received a command
  flushLed(10);

  if (Serial.available() > 0) {

    // read the oldest byte in the serial buffer:
   incomingByte = Serial.read();

   // Drive will actually be 110 on Handy Board becuase it
   // inverts digital ins
   if( incomingByte == 'D' ) {
      digitalWrite(bit0, LOW);
      digitalWrite(bit1, LOW);
      digitalWrite(bit2, HIGH);
   }
   // Left (101 on Handy)
   else if( incomingByte == 'L' ) {
      digitalWrite(bit0, LOW);
      digitalWrite(bit1, HIGH);
      digitalWrite(bit2, LOW);
   }
   // Right ( 100 on Handy )
   else if( incomingByte == 'R' ) {
      digitalWrite(bit0, LOW);
      digitalWrite(bit1, HIGH);
      digitalWrite(bit2, HIGH);
   }
   // Stop (011 on Handy)
   else if( incomingByte == 'S' ) {
      digitalWrite(bit0, HIGH);
      digitalWrite(bit1, LOW);
      digitalWrite(bit2, LOW);
   }
   // Speak (010 on Handy)
   else if( incomingByte == 'T' ) {
      digitalWrite(bit0, HIGH);
      digitalWrite(bit1, LOW);
      digitalWrite(bit2, HIGH);
   }
   // reverse (001 on H
   else if( incomingByte == 'E' ) {
      digitalWrite(bit0, HIGH);
```

```
            digitalWrite(bit1, HIGH);
            digitalWrite(bit2, LOW);
          }
        }

        // allow the handy board 100 ms to read the signal, then change it to null (111)
        delay(100);
        digitalWrite(bit0, HIGH);
        digitalWrite(bit1, HIGH);
        digitalWrite(bit2, HIGH);
        }
       }
      }
```

Appendix C – HandyBoard IC Code

```
/* Robotic Vehicle Command Control
*  Robot does different functions depending on the state of three digital input pins
*/
#define  EncoderCountsPerRev (12.0)
#define  AdvancePerRev      (5.96458)  //Calibrated wheel circumferance in inches
#define  EstimatedRevTime  (2000.0)  // (over)estimated time to do a revolution of
the wheel in milliseconds
#define  EncoderMotorLeft  (0)
#define  EncoderMotorRight  (1)
#define  GAIN (2)
#define  motorLeft (3)
#define  motorRight (0)
#define  MotorStallPower (8)
#define  CCWDistancePer90Turn (8.4)      // counter clockwise Turning distance
#define  CWDistancePer90Turn (8.4)      // clockwise turning distance
#define  TurningPowerLock  (-5)      //motor power of the opposite wheel in order to
lock it
#define  ClockWise (1)            //int initializeEncoders(int a,int b);
//int drive(float a);

// bumpers
#define LEFTTOUCH  (digital(15))
#define RIGHTTOUCH (digital(14))

// Arduino to Handy Board Action Command Pins
#define BIT0 (digital(10))
#define BIT1 (digital(11))
#define BIT2 (digital(12))

#define DRIVE (6)
#define LEFT (5)
#define RIGHT (4)
```

```
#define STOP (3)
#define SPEAK (2)

#define STOP_TURN (15.0)
#define TURN (30.0)
#define DRIVE_SPEED (50)

int drive_state = 0;
int motor_left_speed = 0;
int motor_right_speed = 0;

void main()
{
  //int counter = 5;
  float distance; // how far we want to go
  while(!start_button());

  /*motor(0,50);

  while(!stop_button())
   {
    //counter--;
    motor(3,-counter);
    sleep(2.0);
    printf("/n%d",counter);
  }
  */

  initializeEncoders(EncoderMotorLeft,EncoderMotorRight);

  // Drive = 110 = 6
  // Left = 101 = 5
  // Right = 100 = 4
  // Stop = 011 = 3
  // Speak = 010 = 2

  while(!stop_button()) {
    int code = BIT0 || (BIT1>>1) || (BIT2>>2);
    printf("%d\n",code);
    if( BIT0 == 1 && BIT1 == 1 && BIT2 == 0 ) {
      printf("DRIVE\n");
      drive_state = 1;
      motor_right_speed = DRIVE_SPEED;
      motor_left_speed = DRIVE_SPEED;
      motor(motorLeft, motor_left_speed);
      motor(motorRight, motor_right_speed);
      sleep(0.5);
    }
    else if( BIT0 == 1 && BIT1 == 0 && BIT2 == 1 ) {
      printf("LEFT\n");
```

```
    if( drive_state == 0 ) {
      turn(!ClockWise, STOP_TURN);
    }
    else if( drive_state == 1 ) {
      turn(!ClockWise, TURN);
      motor(motorLeft, motor_left_speed);
      motor(motorRight, motor_right_speed);
    }
    else if( drive_state == 2 ) {
      reverseTurn(!ClockWise, TURN);
      motor(motorLeft, motor_left_speed);
      motor(motorRight, motor_right_speed);
    }
  }
  else if( BIT0 == 1 && BIT1 == 0 && BIT2 == 0 ) {
    printf("RIGHT\n");
    if( drive_state == 0 ) {
      turn(ClockWise, STOP_TURN);
    }
    else if( drive_state == 1 ) {
      turn(ClockWise, TURN);
      motor(motorLeft, motor_left_speed);
      motor(motorRight, motor_right_speed);
    }
    else if( drive_state == 2 ) {
      reverseTurn(ClockWise, TURN);
      motor(motorLeft, motor_left_speed);
      motor(motorRight, motor_right_speed);
    }
  }
  else if( BIT0 == 0 && BIT1 == 1 && BIT2 == 1 ) {
    printf("STOP\n");
    stallWithState();
    drive_state = 0;
  }
  else if( BIT0 == 0 && BIT1 == 1 && BIT2 == 0 ) {
    printf("SPEAK\n");
    stallWithState();
    tone(264.0, 0.2);
    tone(444.0, 0.6);
    tone(264.0, 0.2);
    tone(444.0, 0.4);
    tone(444.0, 0.2);
  }
  else if( BIT0 == 0 && BIT1 == 0 && BIT2 == 1 ) {
    printf("REVERSE\n");
    drive_state = 2;
    motor_right_speed = -DRIVE_SPEED;
    motor_left_speed = -DRIVE_SPEED;
    motor(motorLeft, motor_left_speed);
```

```
        motor(motorRight, motor_right_speed);
        sleep(0.5);
      }
      /*int command = BIT0 | (BIT1 << 1) | (BIT2 << 2)*/
    }

    // drive(1.2);
    // turn(!ClockWise, 90.0);
    //2 while(!start_button());

    disableEncoders(EncoderMotorLeft,EncoderMotorRight);

    ao();
}


// Stall the car wether it is driving or reversing
void stallWithState() {
    if( drive_state == 1 ) {
        stallMotorWhileDriving();
    }
    else if( drive_state == 2 ) {
        stallMotorWhileReversing();
    }
}

//////////
int initializeEncoders(int ch1,int ch2)
{
    enable_encoder(ch1);
    enable_encoder(ch2);
    return(0);
}

int disableEncoders(int ch1,int ch2)
{
    disable_encoder(ch1);
    disable_encoder(ch2);
    return(0);
}

// stalls the motor by running it at a speed that will stop it
void stallMotorWhileDriving()
{
    motor(motorLeft, -MotorStallPower);
    motor(motorRight, -MotorStallPower);
    sleep(.2);
    ao();
}
```

```
// stalls the motor by running it at a speed that will stop it
void stallMotorWhileReversing()
{
   motor(motorLeft, MotorStallPower);
   motor(motorRight, MotorStallPower);
   sleep(.2);
   ao();
}

/*
 * Reverse()
 *
 * Drives the robot backwards a certain exact distance
 * Uses an encoder on each motor to make sure the robot travels the desired distance
 * By counting the number of revolutions of the wheel
 */
int reverse(float distance)
{
   int countsDesired, countsCompleted;    //Variables to store the encoder counts and
compute the error
   long estimatedBackupTime, startTime;
   int cmdLeft, cmdRight;
   int countLeft = 0;
   int countRight = 0;
   int errorLeft, errorRight;
   int power;

   // find (over)estimated back up time
   estimatedBackupTime = (long)(EstimatedRevTime * (distance / AdvancePerRev));

   countsDesired = (int) ((distance / AdvancePerRev) * EncoderCountsPerRev);
   reset_encoder(EncoderMotorLeft);
   reset_encoder(EncoderMotorRight);
   errorLeft = 1;
   errorRight =  1;

   startTime = mseconds();

   // loop until the correct distance has been traveled
   while(errorLeft > 0 || errorRight > 0)
    {
      cmdLeft = computeCommand(GAIN, errorLeft);
      cmdRight = computeCommand(GAIN, errorRight);
      motor(motorLeft, -cmdLeft);
      motor(motorRight, -cmdRight);
      countLeft = read_encoder(EncoderMotorLeft);
      countRight = read_encoder(EncoderMotorRight);
      errorLeft = countsDesired - countLeft;
      errorRight = countsDesired - countRight;
```

```
      // check for infinite backup stuck
      if( mseconds() - startTime > estimatedBackupTime ) {
         errorLeft = -1;
         errorRight = -1;
      }

      printf("eL= %d, eR= %d,  cL=%d, cR=%d\n",errorLeft, errorRight, cmdLeft,
cmdRight);
   }
   ao();    //turn the motors off
}

/*
 * Turn()
 * Turns the robot a certain degrees in a certain direction
 *
 */
void turn(int clockwise, float degrees)
{

   int countsDesired, countsCompleted;    //Variables to store the encoder counts and
compute the error
   int cmdLeft=100, cmdRight=100;
   int countLeft = 0;
   int countRight = 0;
   int errorLeft, errorRight;
   int power;
   float DistancePer90Turn = (degrees/90.0) * CWDistancePer90Turn;

   // CCW distance is different than CW distance switch if appropriate
   if( !clockwise ) {
      DistancePer90Turn = (degrees/90.0) * CCWDistancePer90Turn;
   }

   countsDesired = (int) ((DistancePer90Turn / AdvancePerRev) *
EncoderCountsPerRev);
   reset_encoder(EncoderMotorLeft);
   reset_encoder(EncoderMotorRight);
   errorLeft = 1;
   errorRight =  1;

   while((errorRight > 0) && !clockwise)
    {
      motor(motorRight, cmdRight);
      motor(motorLeft, TurningPowerLock);

      countRight = read_encoder(EncoderMotorRight);
      errorRight = countsDesired - countRight;

      // check for bumper hits
```

```
    if( LEFTTOUCH == 1 ) {
      // stall motor
      ao();

      // save encoder state and call reverse function
      countsDesired = errorRight;
      reverse(AdvancePerRev/4.0);
      reset_encoder(EncoderMotorLeft);
      reset_encoder(EncoderMotorRight);
      // printf("Left\n");
    }
    else if( RIGHTTOUCH == 1 ) {
      // stall motor
      ao();

      // save encoder state and call reverse function
      countsDesired = errorRight;
      reverse(AdvancePerRev/4.0);

      reset_encoder(EncoderMotorLeft);
      reset_encoder(EncoderMotorRight);
      // printf("Right\n");
    }

    printf("eL= %d, eR= %d,  cL=%d, cR=%d\n",errorLeft, errorRight, cmdLeft,
cmdRight);
  }

  while(errorLeft > 0 && clockwise)
   {
    //cmdLeft = computeCommand(GAIN, errorLeft);

    motor(motorLeft, cmdLeft);
    motor(motorRight, TurningPowerLock);
    countLeft = read_encoder(EncoderMotorLeft);
    errorLeft = countsDesired - countLeft;

    // check for bumper hits
    if( LEFTTOUCH == 1 ) {
      // stall motor
      ao();

      // save encoder state and call reverse and turn functions
      countsDesired = errorLeft;
      reverse(AdvancePerRev/4.0);
      reset_encoder(EncoderMotorLeft);
      reset_encoder(EncoderMotorRight);
      printf("Left\n");
    }
    else if( RIGHTTOUCH == 1 ) {
```

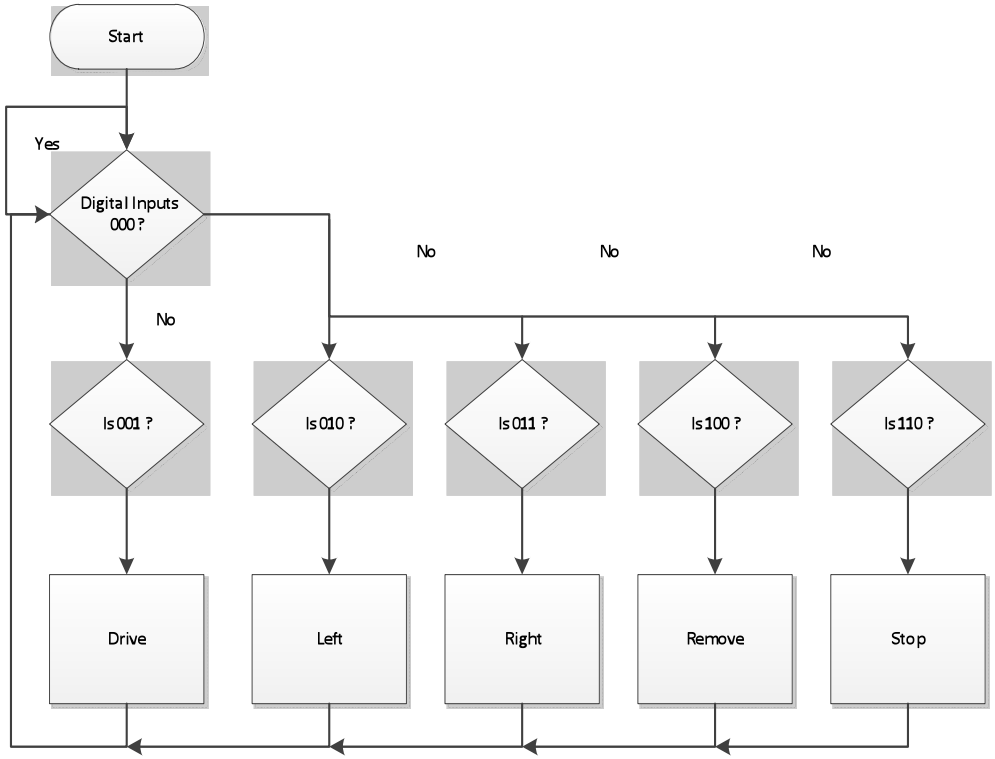
```
        // stall motor
        ao();

        // save encoder state and call reverse and turn functions
        countsDesired = errorLeft;
        reverse(AdvancePerRev/4.0);

        reset_encoder(EncoderMotorLeft);
        reset_encoder(EncoderMotorRight);
        printf("Right\n");
      }

     printf("eL= %d, eR= %d,  cL=%d, cR=%d\n",errorLeft, errorRight, cmdLeft,
cmdRight);

  }
  ao();
}


/*
 * Turn()
 * Turns the robot a certain degrees in a certain direction if it is going in reverse
 *
 */
void reverseTurn(int clockwise, float degrees)
{

  int countsDesired, countsCompleted;    //Variables to store the encoder counts and
compute the error
  int cmdLeft=100, cmdRight=100;
  int countLeft = 0;
  int countRight = 0;
  int errorLeft, errorRight;
  int power;
  float DistancePer90Turn = (degrees/90.0) * CWDistancePer90Turn;

  // CCW distance is different than CW distance switch if appropriate
  if( !clockwise ) {
    DistancePer90Turn = (degrees/90.0) * CCWDistancePer90Turn;
  }

  countsDesired = (int) ((DistancePer90Turn / AdvancePerRev) *
EncoderCountsPerRev);
  reset_encoder(EncoderMotorLeft);
  reset_encoder(EncoderMotorRight);
  errorLeft = 1;
  errorRight =  1;

  while((errorRight > 0) && !clockwise)
```

```
    {
      motor(motorRight, -cmdRight);
      motor(motorLeft, -TurningPowerLock);

      countRight = read_encoder(EncoderMotorRight);
      errorRight = countsDesired - countRight;

      // check for bumper hits
      if( LEFTTOUCH == 1 ) {
        // stall motor
        ao();

        // save encoder state and call reverse function
        countsDesired = errorRight;
        reverse(AdvancePerRev/4.0);
        reset_encoder(EncoderMotorLeft);
        reset_encoder(EncoderMotorRight);
        // printf("Left\n");
      }
      else if( RIGHTTOUCH == 1 ) {
        // stall motor
        ao();

        // save encoder state and call reverse function
        countsDesired = errorRight;
        reverse(AdvancePerRev/4.0);

        reset_encoder(EncoderMotorLeft);
        reset_encoder(EncoderMotorRight);
        // printf("Right\n");
      }

      printf("eL= %d, eR= %d,  cL=%d, cR=%d\n",errorLeft, errorRight, cmdLeft,
cmdRight);
    }

    while(errorLeft > 0 && clockwise)
     {
      //cmdLeft = computeCommand(GAIN, errorLeft);

      motor(motorLeft, -cmdLeft);
      motor(motorRight, -TurningPowerLock);
      countLeft = read_encoder(EncoderMotorLeft);
      errorLeft = countsDesired - countLeft;

      // check for bumper hits
      if( LEFTTOUCH == 1 ) {
        // stall motor
        ao();
```

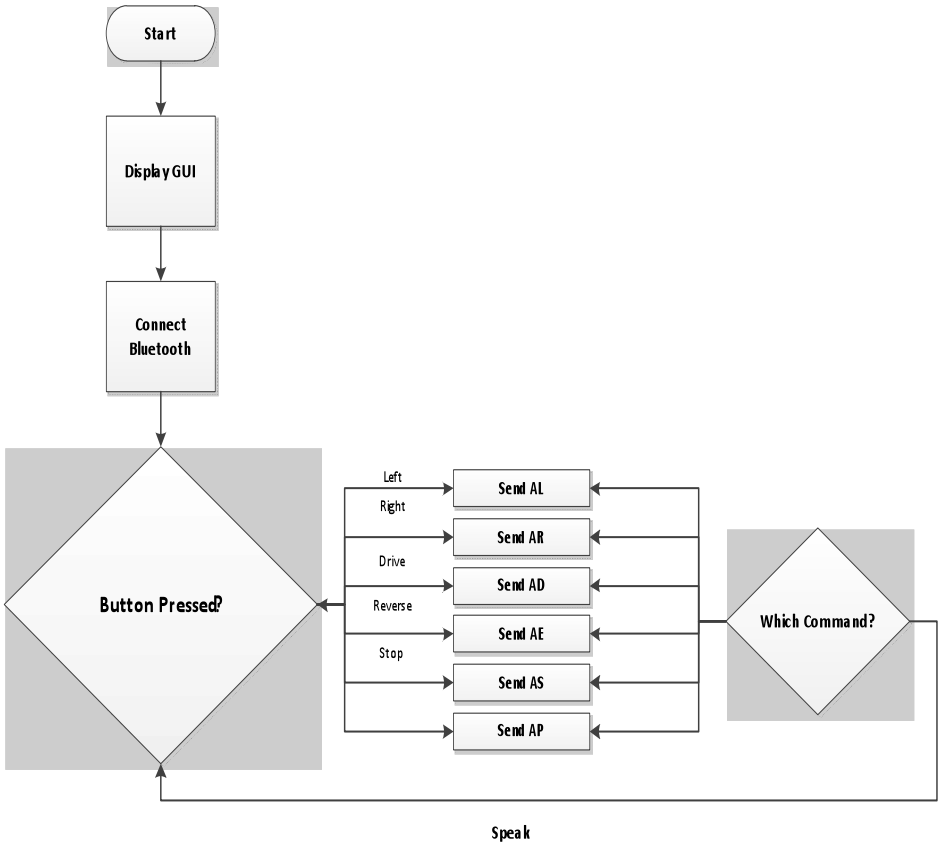
```
        // save encoder state and call reverse and turn functions
        countsDesired = errorLeft;
        reverse(AdvancePerRev/4.0);
        reset_encoder(EncoderMotorLeft);
        reset_encoder(EncoderMotorRight);
        printf("Left\n");
      }
      else if( RIGHTTOUCH == 1 ) {
        // stall motor
        ao();

        // save encoder state and call reverse and turn functions
        countsDesired = errorLeft;
        reverse(AdvancePerRev/4.0);

        reset_encoder(EncoderMotorLeft);
        reset_encoder(EncoderMotorRight);
        printf("Right\n");
      }

      printf("eL= %d, eR= %d,  cL=%d, cR=%d\n",errorLeft, errorRight, cmdLeft,
cmdRight);

    }
    ao();
}


// make sure the motor power levels are in the range:
// 15 to 20 and -15 to -20
int computeCommand(int g, int e)
{
    int cmd = 0;

    cmd = (g * e);

    // power levels less than 15 will stall the motor
    if( cmd > 0 && cmd < 15 ) {
      cmd = 15;
    }
    if( cmd > -15 && cmd < 0 ) {
      cmd = -15;
    }

    // power levels greater than 20 will damage the bumper sensors
    if (cmd  > 100)
     cmd = 100;
    else
     if (cmd < -100)
     cmd = -100;
```

```
    return(cmd);
}
```

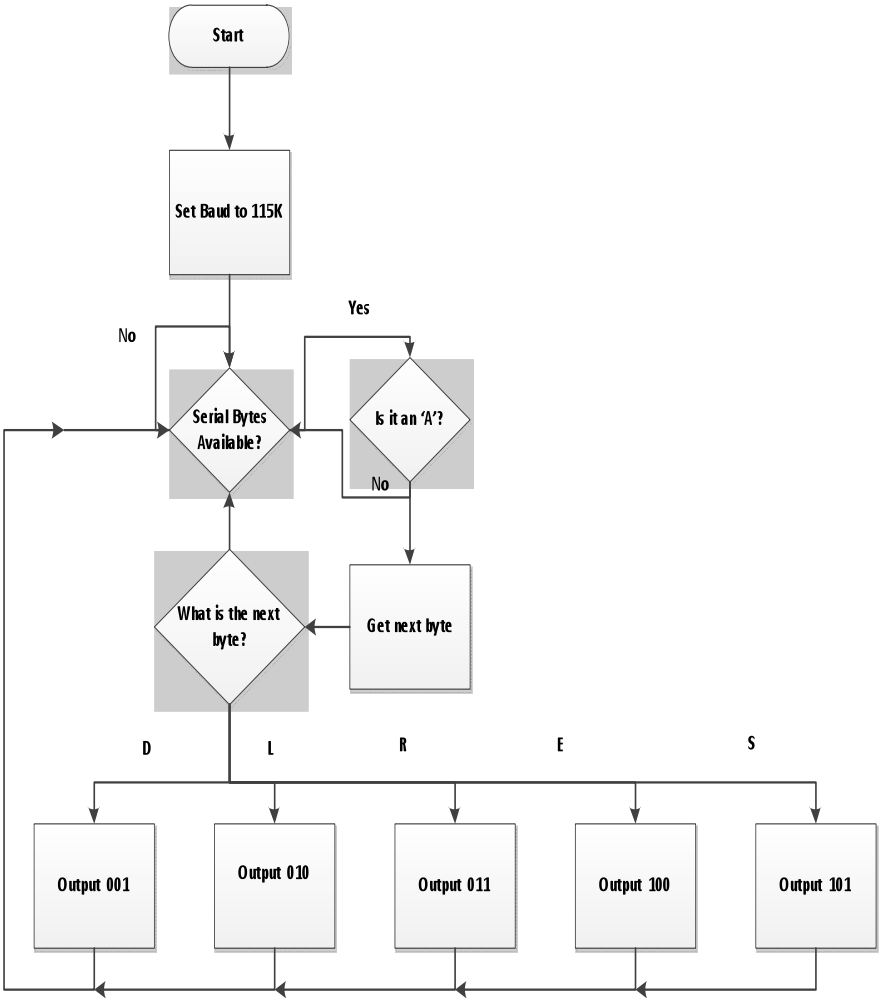
Appendix D-IC Flowchart



Appendix E-Android Flowchart



Appendix F-Arduino Flowchart



References

1. Jon Don't Do It, http://jondontdoit.blogspot.com/
2. Roving Networks Wireless for Less, Advanced User Manual, http://www.sparkfun.com/datasheets/Wireless/Bluetooth/rn-bluetooth-um.pdf
3. Amarino "Android Meets Arduino", http://www.amarino-toolkit.net/index.php/docs.html
4. Sparkfun Electronics, http://www.sparkfun.com/products/9358