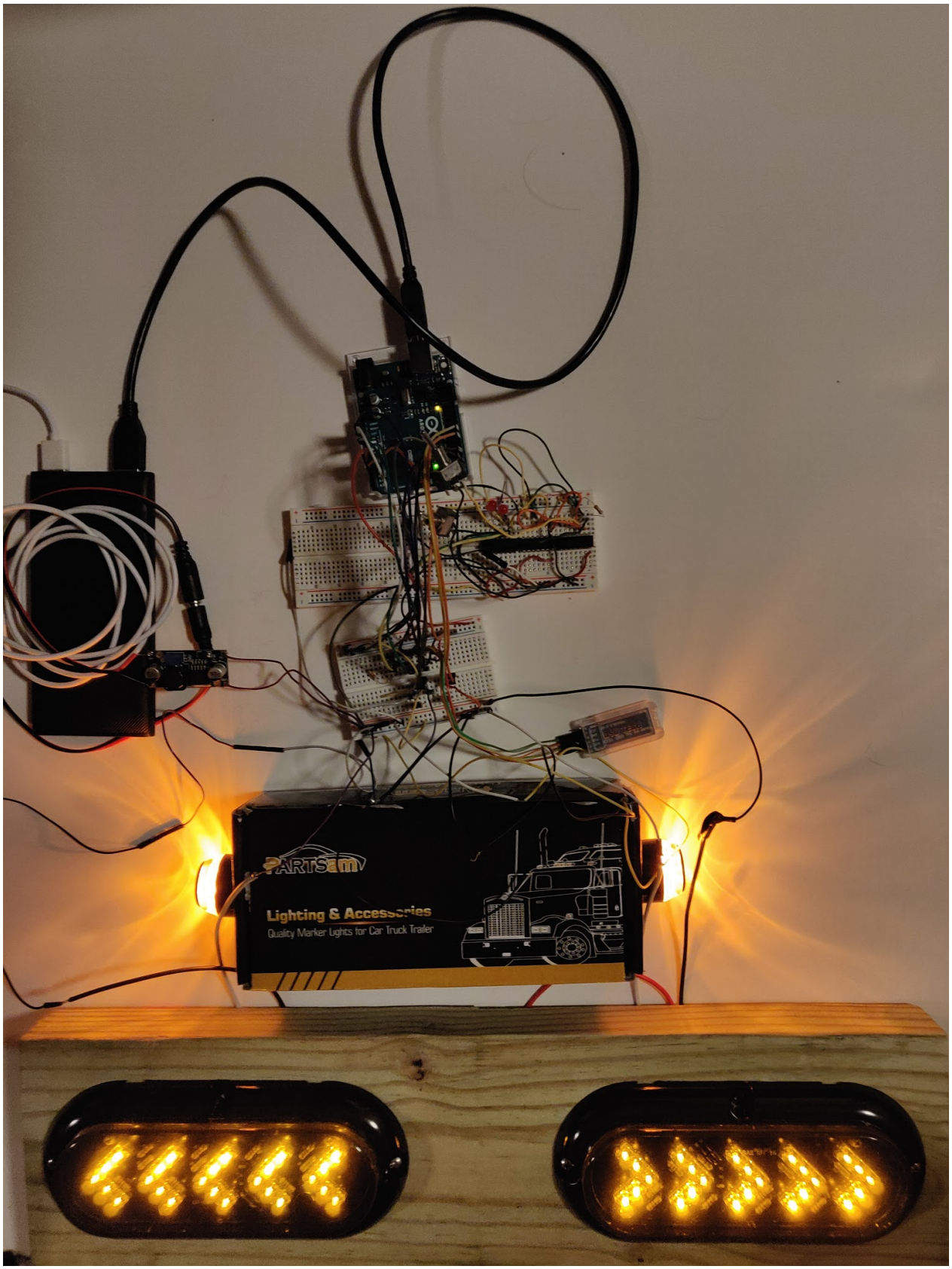

Lighting & Accessories
Quality Marker Lights for Car Truck Trailer

Bike Turn Signal Fall 2019 ECE 110/120 Honors Lab Final Report

Lingxiao Mou (lmou2), Carter Smith (carters3), Dawid Bycul (dbycul2)

Introduction:

On a campus as large and complex as the University of Illinois, it can often be difficult to navigate while riding a bike. Automobiles rarely pay attention for hand signals while turning and finding your way through campus is sometimes near impossible. To solve this, we're proposing a handlebar-mounted system that connects via Bluetooth to Google Maps and indicates when turns need to be made. Furthermore, this system will incorporate a turn signal on the rear of the bike for the purpose of alerting automobiles about upcoming turns. For our project to be considered successful, the following criteria must be met:

1. *The user must be alerted of the intended directions provided by Google Maps.*
2. *The connectivity between our microcontroller and phone must be bluetooth.*
3. *The turn signal must be an independent system, able to run without Google Maps or a seperate phone and must alert vehicles from the rear of the user's intended turn.*
4. *Our entire bike system should be battery powered, preferably rechargeable.*

All in all, our system provides the user with a system to navigate using Google Maps without having to look at their phone or use audio assistance and a way to notify rear vehicles about the user's intention of turning. This increases the safety of bicycle riders by increasing communication on the road and decreasing distractions and hazards surrounding looking at a phone while riding on a bike.

Design

Components

1. Characteristics of Sensors and Logic Component

   A. Bluetooth module (HC-05)

Pin num	Pin name	Description
1	Enable/Key	This pin is used to toggle between Data Mode (set low) and AT command mode (set high). By default it is in Data mode
2	Vcc	Connect to Power, we used 5V supply to from the Arduino
3	Ground	Ground pin of module, we have it connect to ground of Arduino
4	TX Transmitter	Transmits Serial Data. Everything received via Bluetooth will be given out by this pin as serial data. We have it connect to corresponding pins in Arduino to transmit LEFT and RIGHT signal to serial data.
5	RX-receiver	Receive Serial Data. Every serial data given to this pin will be broadcasted via Bluetooth. We have it connect to corresponding pins in Arduino so that it receives data from people's phone app.

Overall, the Bluetooth module serves as the transition station since it collects data from user's Google Map application and lights up the corresponding LEDs at the front of the car. After testing and online researching, we figured out that the Data mode Baud rate is 9600, the operating voltage is from 4 to 6 volts (we use 5 volts), the operating current is 30mA and the range of signal is around 100 meters.

B. Logic gates

a. Truth table

Bit pattern	**Output**	**Meaning**
00	0	No LED should be turned on
01	1	Left LED should be on
10	1	Right LED should be on
11	0	Both LED are off in case of improper user input, which means both LEDs are turned on at the same time

b. K-map

Left switch/Right switch	0	1
0	0	1
1	1	0

Overall, the XOR gate is used to prevent improper usage by the rider. We connect the XOR gate between the switches on the handlebar of bike (since we have not install the product to the bike, XOR will only be connected with two switches) and the two ellipse LEDs.

2. Design Analysis

a. Bluetooth module

The Bluetooth module helps us receives the “LEFT” and “RIGHT” signal from the phone app and turn on and off the LEDs at the handle bar. Therefore, the Bluetooth module and Arduino should be placed at the front of the bike to provide signal to those front LEDs. This allows for a more robust and seamless design, an aspect crucial for designing a consumer product.

b. Gate Combination(XOR,NAND,NOT)

Since the gates has to be implemented on breadboard, we put the XOR, NAND, NOT gate on breadboard and then use two wires to connect the arduino input A1 and A0 , which connects to the the two ellipse LEDs at the back of the bike.

Results and Future Plans

1. Turn Indicator Demo (Link to switch demo and early testing : shorturl.at/fwST2 shorturl.at/cjyCR)

2. In the end, our design follows the criteria that were established in the introduction, with pending functionality for the app to work in the background, a mute switch to ignore the automatic notifications, 3-D printed fittings for a bike, and a casing for the final form factor which would include a smaller microcontroller, battery, and printed PCB.

Problems and Challenges

Conclusion:

1. Lessons Learned

We first had problems of finding the suitable Bluetooth module. At the beginning, we chose the BBC Micro:bit since we saw that it has a corresponding application on phone and is easy to program. However, it has really poor Bluetooth pairing capability since it can rarely successfully connect with phone. As a result, we have to change to another Bluetooth module since it failed to work. We learned that thorough investigation about components is necessary. This also taught us that it is crucial to be able to adapt a prototype when different conditions arise. Otherwise, it will just cause delay to the final product. Additionally, we were having problems with keeping power bank on. Since the power bank we bought will only supply 5 volts and our Arduino cannot provide enough amperage for power bank to turn on for a long time. The current draw from the Arduino is miniscule compared to the devices that the power bank expects to charge, and, as a result, the powerbank does not recognize the Arduino as a connected device. The solution for this problem is that we connect the power bank with two LEDs that are always on to extract power from power bank. As a result, the power bank can stay on to supply all of our work. We also had success with keeping the power bank on by having a device connected to the Arduino through Bluetooth. Moreover, we had problems with receiving Bluetooth signal from phone to Arduino. The solution to this problem is that an end sentence character is necessary to align the same format between signal and receiver Arduino. Smaller, less lengthy problems included finding out about pull-down resistors, wiring and finding the correct transistor, and finding code online to help us connect our phone through an app to an Arduino over Bluetooth and send a serial signal from our phone app. Overall, we learned that thorough investigation of a product is necessary, including the format of output, online review and parameter of product. We also discovered how useful the Bluetooth module is; it proved critical to connecting the user's phone to the turn indicators in a seamless fashion.

Self Assessment

Overall, our final project has met all of the requirements specified in the introduction. Our product can successfully alert user by directions sent from Google Map, connect microcontroller and phone with bluetooth, have seperate user-alerting and surround alerting system, and can be fully powered with a portable and rechargeable battery bank.

References

[1]"Arduino turn by turn navigation system", *Forum.arduino.cc*, 2019. [Online]. Available: https://forum.arduino.cc/index.php?topic=256568.0. [Accessed: 14- Dec- 2019].

[2]"CD40106B CMOS Hex Schmitt-Trigger Inverters", *Texas Instruments*, 2017. [Online]. Available: http://www.ti.com/lit/ds/symlink/cd40106b.pdf. [Accessed: 14- Dec- 2019].

[3]"Quadruple 2-Input Exclusive-OR Gates", *Media.digikey.com*, 2010. [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Texas%20Instruments%20PDFs/SN54,74(LS,S)86(A).pdf. [Accessed: 14- Dec- 2019].

[4]"CD40106B CMOS Hex Schmitt-Trigger Inverters", *Texas Instruments*, 2019. [Online]. Available: http://www.ti.com/lit/ds/symlink/cd40106b.pdf. [Accessed: 14- Dec- 2019].

Appendix

General Diagram:

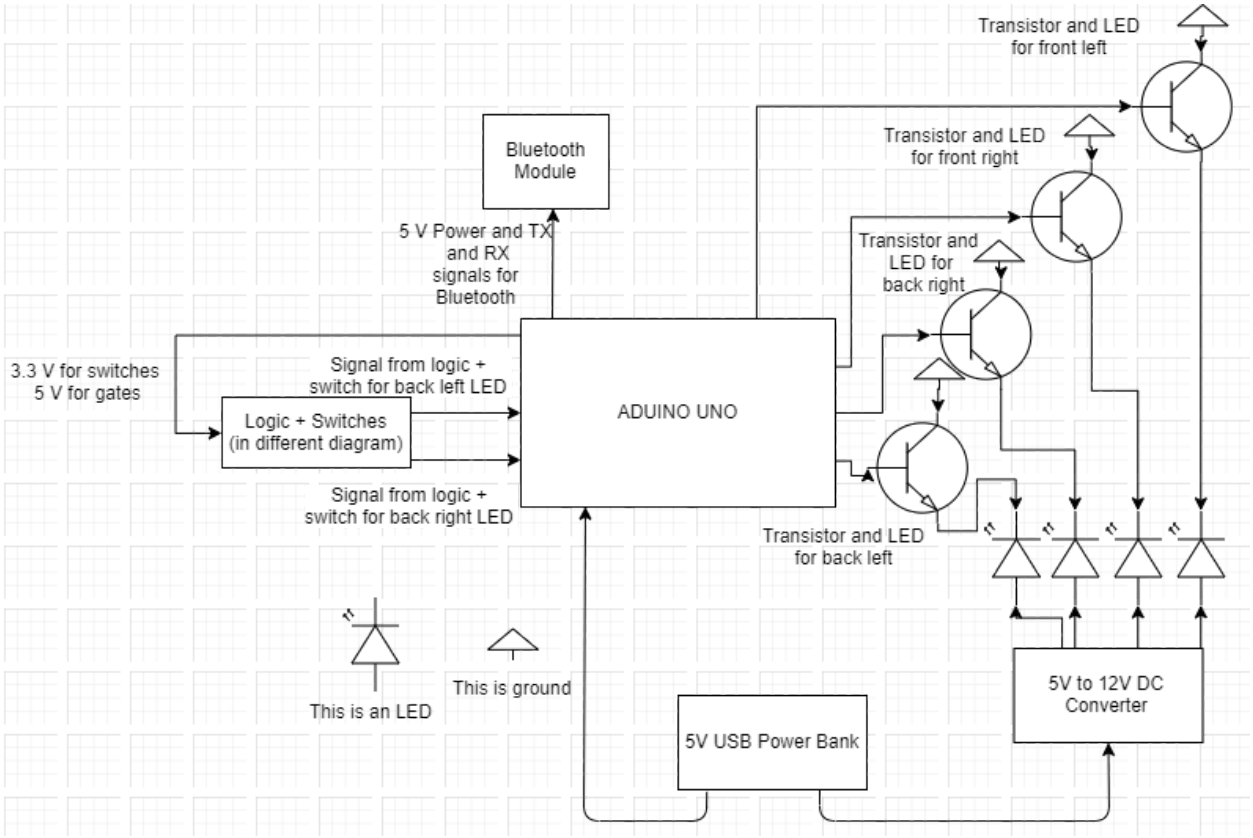
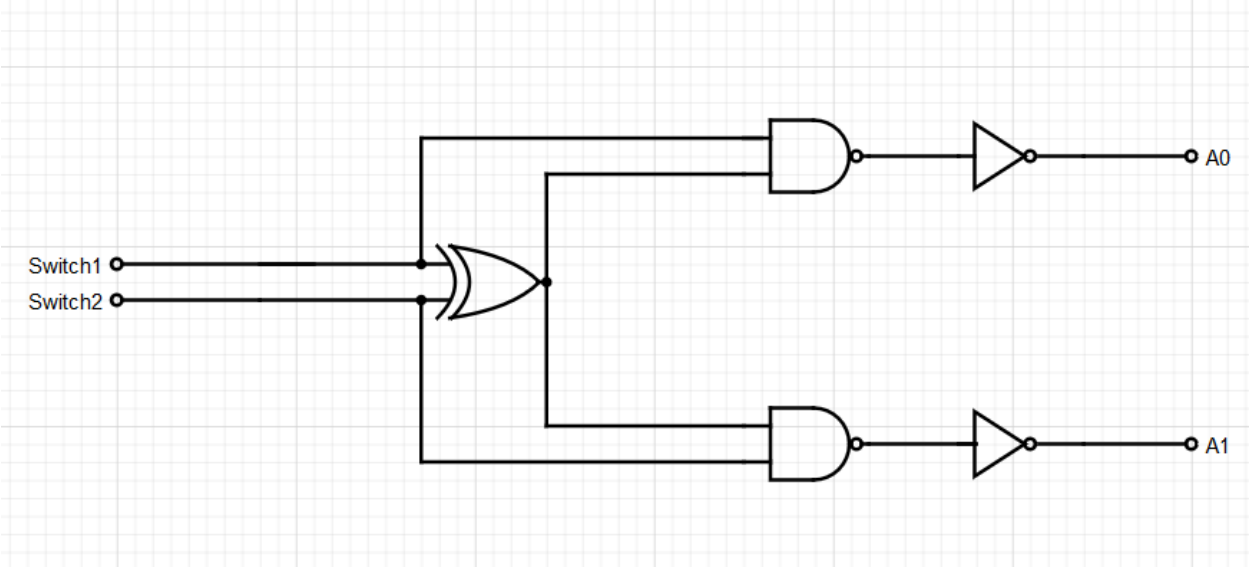


Diagram for Logic and Switches:



Code for Android Application:

Notification Service to read the arrow type from the Google Maps Notification Bar:

```
package com.example.bicycleturnsignals;


import android.app.Notification;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.os.IBinder;
import android.provider.MediaStore;
import android.service.notification.NotificationListenerService;
import android.service.notification.StatusBarNotification;
import android.util.Log;
import android.view.View;
```

```
import java.util.Arrays;

//import androidx.localbroadcastmanager.content.LocalBroadcastManager;

public class NotificationService extends NotificationListenerService {

    Context context;

    @Override

    public void onCreate() {

        super.onCreate();
        context = getApplicationContext();

    }
    @Override

    // This is fired whenever a new notification appears. For google maps navigation this happens
about every second
    // Most of the code in this function is just to print all the data to logcat
    public void onNotificationPosted(StatusBarNotification sbn) {
        // Test if the notification is from google maps
        String pack = sbn.getPackageName();
        if (!pack.equals("com.google.android.apps.maps")) {
            return;
        }
        // Some string
        String ticker = "";
        if (sbn.getNotification().tickerText != null) {
            ticker = sbn.getNotification().tickerText.toString();
        }
        // Extras is a bundle that contains basically all of the notification data
        Bundle extras = sbn.getNotification().extras;
        if (extras.keySet() != null) {
            // Print all the variables in extras
            Object[] keys = extras.keySet().toArray();
            for (int i = 0; i < keys.length; i++) {
                if (keys[i] instanceof String && extras.get((String) keys[i]) != null) {
                    //Log.i((String) keys[i], extras.get((String) keys[i]).toString());
                }
            }
         //   Log.i("KEYSET", extras.keySet().toString());
        }
```

```
if ( extras.getString("android.subText") != null) {
   // Log.i("SUBTEXT", extras.getString("android.subText"));
}
//View cv = (View) extras.get("android.contains.customView");
//cv.getSourceLayoutResId();
String title = extras.getString("android.title");
String text = "";
if (extras.getCharSequence("android.text") != null) {
   text = extras.getCharSequence("android.text").toString();
}

// Useless
//Log.i("AX", "getting icon");
int iconId = extras.getInt(Notification.EXTRA_LARGE_ICON_BIG);
//Log.i("ICON", String.valueOf(iconId));

// Get the arrow icon
if (extras.containsKey(Notification.EXTRA_LARGE_ICON)) {
   Bitmap bitmap = (Bitmap) extras.get(Notification.EXTRA_LARGE_ICON);
   // Calculate a hash that uniquely identifies the arrow image
   if (bitmap == null) {
      return;
   }
   String hash = String.valueOf(bitmapHash(bitmap));

   // Send that hash to the main activity to show the arrow in the app (You don't need that)
   SharedPreferences prefs = getSharedPreferences("arrowInts", 0);
   if (prefs.getInt(hash, 5) == 5) {
      new ImageSaver(getApplicationContext())
            .setFileName(hash)
            .save(bitmap);
      prefs.edit().putInt(hash, Directions.Companion.getUNKNOWN()).apply();
   }

   int distance = extractDistance(title);

   Intent intent = new Intent();
   intent.setAction("com.example.bicycleturnsignals");
   intent.putExtra("imgHash", hash);
   intent.putExtra("distance", distance);
   sendBroadcast(intent);

   //Log.i("IMG HASH", String.valueOf(bitmapHash(bitmap)));
}
```

```
        //Log.i("Package",pack);
        //Log.i("Ticker",ticker);
        if (title != null) {
            Log.i("Title", title);
        }
        //Log.i("Text",text);

        Intent msgrcv = new Intent("Msg");
        msgrcv.putExtra("package", pack);
        msgrcv.putExtra("ticker", ticker);
        msgrcv.putExtra("title", title);
        msgrcv.putExtra("text", text);

        //LocalBroadcastManager.getInstance(context).sendBroadcast(msgrcv);


    }

    private int extractDistance(String title) {
        if (title.indexOf("ft") > 0) {
            return Integer.valueOf(title.split(" ", 2)[0]);
        }
        return -1;
    }

    private int bitmapHash(Bitmap bitmap) {
        int width = bitmap.getWidth();
        int height = bitmap.getHeight();
        int[] buffer = new int[width * height];
        bitmap.getPixels(buffer, 0, width, 0, 0, bitmap.getWidth(), bitmap.getHeight());
        return Arrays.hashCode(buffer);
    }

    @Override

    public void onNotificationRemoved(StatusBarNotification sbn) {
        Log.i("Msg","Notification Removed");

    }
}
```

Main code to send Bluetooth signal and to connect to Arduino :

```
package com.example.bicycleturnsignals
```

```
import android.app.ProgressDialog
import android.bluetooth.BluetoothAdapter
import android.bluetooth.BluetoothSocket
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.content.Intent
import android.content.BroadcastReceiver
import android.content.Context
import androidx.core.content.ContextCompat.getSystemService
import android.icu.lang.UCharacter.GraphemeClusterBreak.T
import android.util.Log
import android.content.IntentFilter
import androidx.core.content.ContextCompat.getSystemService
import android.icu.lang.UCharacter.GraphemeClusterBreak.T
import androidx.core.content.ContextCompat.getSystemService
import android.icu.lang.UCharacter.GraphemeClusterBreak.T
import android.view.LayoutInflater
import androidx.core.content.ContextCompat.getSystemService
import android.icu.lang.UCharacter.GraphemeClusterBreak.T
import android.view.View
import android.widget.*
import androidx.core.app.ComponentActivity.ExtraData
import androidx.core.content.ContextCompat.getSystemService
import android.icu.lang.UCharacter.GraphemeClusterBreak.T
import android.os.AsyncTask
import java.io.IOException
import java.util.*


class MainActivity : AppCompatActivity() {
    // BLUETOOTH
    internal var myBluetooth: BluetoothAdapter? = null
    internal var btSocket: BluetoothSocket? = null
    private var isBtConnected = false
    internal val myUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB")
    internal var address: String? = "00:14:03:06:1F:9A"
    private var progress: ProgressDialog? = null
    // END BLUETOOTH

    private var direction: Int = 4

    var broadcastReceiver: BroadcastReceiver = object : BroadcastReceiver() {
        override fun onReceive(context: Context, intent: Intent) {
            val s1 = intent.getStringExtra("imgHash")
```

```
            val distance = intent.getIntExtra("distance", -1)
            Log.i("NEW IMAGE", s1)
            loadArrow(s1, false)
            var newDirection = getSharedPreferences("arrowInts", 0).getInt(s1, 4)
            Log.d("DIRECTION", newDirection.toString())
            if (direction != newDirection && distance != -1 && distance <= 300) {
                direction = newDirection
                sendTurn()
            }
        }
    }

    private fun sendTurn() {
        var cmd = ""
        when(direction) {
            Directions.LEFT -> cmd = "LEFT\r\n"
            Directions.RIGHT -> cmd = "RIGHT\r\n"
            Directions.STRAIGHT -> cmd = "OFF\r\n"
            Directions.UNKNOWN -> cmd = "BOTH\r\n"
            Directions.BACK -> cmd = "BOTH\r\n"
        }
        sendSignal(cmd)
    }

    private fun loadArrow(hash: String, force: Boolean) {
        val imgSaver = ImageSaver(applicationContext)
        val prefs = getSharedPreferences("arrowInts", 0)

        if (prefs.getInt(hash, 5) != 5 && !force) {
            Log.d("LOADARROW", "EXIT"+force.toString())
            return
        }
        if (!force) {
            prefs.edit().putInt(hash, 4).apply()
        }
        Log.d("LOADARROW", "LOADING")

        val parent = findViewById<LinearLayout>(R.id.arrows)
        val inflater = LayoutInflater.from(applicationContext)
        val chunk = inflater.inflate(R.layout.chunk_arrow, parent, false)

chunk.findViewById<ImageView>(R.id.arrow).setImageBitmap(imgSaver.setFileName(hash).l
oad())
        chunk.findViewById<ImageView>(R.id.arrow).setOnClickListener {
            sendSignal("LEFT\r\n")
```

```
        }
        val spinner = chunk.findViewById<Spinner>(R.id.direction)
        ArrayAdapter.createFromResource(
            applicationContext,
            R.array.directions,
            android.R.layout.simple_spinner_item
        ).also { adapter ->
            // Specify the layout to use when the list of choices appears
            adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
            // Apply the adapter to the spinner
            spinner.adapter = adapter
        }
        spinner.setSelection(prefs.getInt(hash, 4))
        spinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
            override fun onNothingSelected(p0: AdapterView<*>?) {
                TODO("not implemented") //To change body of created functions use File | Settings |
File Templates.
            }

            override fun onItemSelected(p0: AdapterView<*>?, p1: View?, position: Int, p3: Long) {
                prefs.edit().putInt(hash, position).apply()
            }
        }
        Log.d("ASDA","Adskada")

        parent.addView(chunk)
    }

    override fun onStart() {
        super.onStart()
        val intentFilter = IntentFilter()
        intentFilter.addAction("com.example.bicycleturnsignals")
        registerReceiver(broadcastReceiver, intentFilter)
        /*val startIntent = Intent(this@MainActivity, BluetoothService::class.java)
        startIntent.action = "om.example.bluetoothservice.action.startforeground"
        startService(startIntent)*/
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        if (getSharedPreferences("arrowInts", 0) != null) {
            val arrows = getSharedPreferences("arrowInts", 0).all
            for ((k, v) in arrows) {
```

```
            Log.d("LOADARROW",k)
            Log.d(k, v.toString())
            loadArrow(k, true)
        }
    }

    findViewById<Button>(R.id.connect).setOnClickListener {
        ConnectBT().execute()
    }


    // BLUETOOTH
    ConnectBT().execute()
}

override fun onStop() {
    super.onStop()
    unregisterReceiver(broadcastReceiver)
    Disconnect()
}


// BLUETOOTH

private fun sendSignal(number: String) {
    var start = System.currentTimeMillis()
    while(System.currentTimeMillis() - start < 100) {}
    if (btSocket != null) {
        try {
            Log.d("SENDING", number)
            btSocket?.getOutputStream()?.write(number.toByteArray())
            //btSocket?.outputStream?.write("RIGHT\r\n".toByteArray(), 0, 7)
        } catch (e: IOException) {
            msg("Error")
        }

    }
}

private fun Disconnect() {
    if (btSocket != null) {
        try {
            btSocket?.close()
        } catch (e: IOException) {
            msg("Error")
```

```
        }

    }

    finish()
}

private fun msg(s: String) {
    Toast.makeText(applicationContext, s, Toast.LENGTH_LONG).show()
}

private inner class ConnectBT : AsyncTask<Void?, Void?, Void?>() {
    private var ConnectSuccess = true

    override fun onPreExecute() {
        progress = ProgressDialog.show(this@MainActivity, "Connecting...", "Please Wait!!!")
    }

    override fun doInBackground(vararg devices: Void?): Void? {
        try {
            if (btSocket == null || !isBtConnected) {
                myBluetooth = BluetoothAdapter.getDefaultAdapter()
                val blue = myBluetooth
                val dispositivo = blue?.getRemoteDevice(address)
                if (dispositivo == null) {
                    Log.d("DISPO", "null")
                }
                btSocket = dispositivo?.createInsecureRfcommSocketToServiceRecord(myUUID)
                val sock = btSocket
                BluetoothAdapter.getDefaultAdapter().cancelDiscovery()
                if (sock == null) {
                    Log.d("SOCK", "null")
                }
                sock?.connect()
            }
        } catch (e: IOException) {
            ConnectSuccess = false
            Log.d("ERROR", e.message)
        }

        return null
    }

    override fun onPostExecute(result: Void?) {
        super.onPostExecute(result)
```

```
            if (!ConnectSuccess) {
                msg("Connection Failed. Is it a SPP Bluetooth? Try again.")
                progress?.dismiss()
                ConnectBT().execute()
                //finish()
            } else {
                msg("Connected")
                isBtConnected = true
            }

            val prog = progress
            prog?.dismiss()
        }
    }
}
```