Dynamic Directions

A magic compass that points to your destination!

By Elizabeth Garner and Joey Horwitz

Demonstration Video

Introduction

Hi! We are Elizabeth Garner and Joey Horwitz, and for our Embeeded OS final project we built a direction finding compass. The project is in two parts: an embedded Raspberry Pi product and an Android app. The user inputs a destination into the app and then the Raspberry Pi controls a physical arrow which points to the target location. This is accomplished using the Google Places API, the phone's GPS module, a magnetometer to measure the Pi's orientation, and a stepper motor to control the arrow. At the end of the day we have a product which allows you to know where you're going while still allowing you to wander. And, once you input the destination, all navigation can be accomplished without staring at a screen. It's ideal for freeform exploration of new places :).

Project Objective:

The goal of our final project is to design a compass that points in the direction of a user-designated destination. As this project rounds out the course, we wanted to incorporate both hardware and software components into our design. As well, we looked to create a unique end product.

Our priority for this project was to process user input with our Raspberry Pi 4 to indicate towards the destination. This allowed us to make design choices in the process for user experience and accuracy. In our final design, we incorporated a motor, motor driver, magnetometer, Raspberry Pi 4, and battery power supply. In terms of software, we developed an app that took destination input, sent current location, and communicated with the Raspberry Pi.

Design

Hardware Design

In our hardware design, we had a stepper motor, motor driver, magnetometer, Raspberry Pi 4, DC power supply, and phone battery power supply. Each of these followed a natural flow as data was received by the Pi and magnetometer, processed by the Pi, passed to the motor driver, and finally displayed with the stepper motor.

For our project, we selected a stepper motor as they are widely used for precise positioning. As well, the stepper motor we found had continuous rotation, allowing for our compass' dial to move as a person holding the compass turns. Some motors on the market have 180 or 360 degrees of rotation. On such motors, the dial must rotate in the incorrect direction, impacting user experience. As well, the stepper motor we used came with a motor driver that was simple to use. We kept the included LEDs that lit up when the motor was turning to alert the user of any changes in direction.

Next, we selected a magnetometer for our design to capture the angle of our compass. Since we could calculate the angle between the current and destination coordinates, we needed to know our angle relative to North to incorporate with this "global" angle. We called our box's angle calculated from the magnetometer our "local" angle. This was important to keep track of at all times as the compass dial would need to compensate for the user not pointing in the same direction as the destination from their current location.

Since magnetometers measure the magnetic field around them, they can pick up noise from surrounding materials. This meant that while we were in the lab, we needed to be careful of what was happening around us and acknowledge that there would be noise. We discussed using a Kalman filter to help with the noise, but ultimately decided not to. We found that the noise within the lab from the nearby projects was less than expected and that the user could still get clear directions from the compass.

Upon first wiring our motor and magnetometer to our Pi, we kept our piTFT for debugging purposes. We had connected our magnetometer to an SPI pin used by the piTFT. In theory, there are multiple channels within this pin to send data in. On the code side, the Pi can look at data being received from a specific channel. This allows for multiple components to be wired to the same channel. However in practice, we were having issues with this. In the code, the channel number must be specified so the Pi knows which channel to be looking at. The channel the magnetometer was sending data over was changing as we were using it. We tried wiring the magnetometer to force it over a default channel, but we were still having issues with receiving the data consistently over this channel. In the end, we decided to remove the piTFT entirely as our final design did not include it.

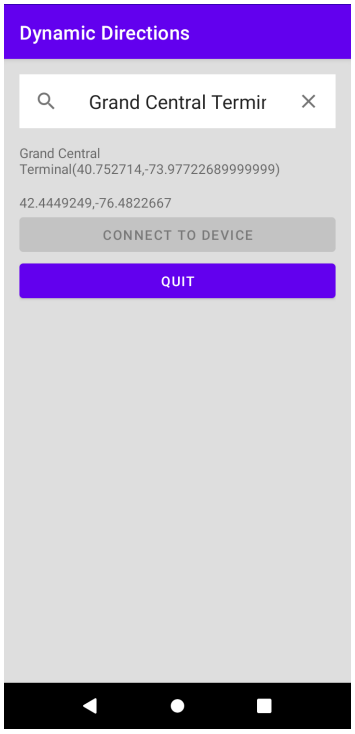
For our power supply, we used two different types of batteries. Motors are notorious for creating issues with power and shorting connected components. This means that we need to be careful to have separate power and ground for our motor. As well, it was important to be able to easily wire in the battery pack. Instead of using a power cable like for the Pi, the motor batteries are wired into the circuitry. Our motor required 5-12V, which the Pi can technically support with its 5V output power pin. However, shorting our Pi with our motor's inconsistent power needs was too risky. For our Pi, we used a phone power bank. The Pi conveniently has a micro USB port for power that regulates power for any surges. This saves our Pi from being shorted from our power supply. A Pi's power pins, however, do not have such power protection.

Software Design

App Design

Socket Programming

An essential aspect of our design included sending and receiving data between our Android app and the Raspberry Pi. Initially, we tried to send information over a Bluetooth connection. Bluetooth does not require a router connection and allows the device to be taken anywhere. We were able to create a



We designed the Android app using Kotlin with Android studio. The main page handles both of our key functions: allowing the user to enter a destination and finding the user's current GPS coordinates. To help us find destination coordinates, we employed the Google Places API and Google Places Autocomplete widget. The autocomplete widget is a pre-built module that offers place suggestions to a user while they type. When the widget is initialized we also initialize a callback function for place selection, so that when a user taps on a desired location the longitude and latitude of that location will be stored for use with the Raspberry Pi. Then, when the Pi initializes a connection and requests destination coordinates, these autocomplete coordinates will be sent over. The app also prints the location name and coordinates to its screen. Both the autocomplete functionality and coordinate fetching require a web connection to the Google Places API, for which we obtained an API key.

To monitor the user's current GPS coordinates we established a thread that works as a repeating timer which polls the GPS once a second. On every timed awakening the thread checks for location permissions, requests the location, and then executes a callback function which both stores the location for TCP and displays the coordinates to the screen.

Aside from these two core functions, we included two buttons in our app. The first was labeled "Connect to Device" and triggered the creation of our TCP server connection. The server is initialized as a runnable class and is told to begin listening. Then, when a request from the Pi comes in, the server will be able to respond with the current and destination locations obtained through the core elements of the app. The other button is called "Quit", and it is used to instruct the Pi to return the compass to its home position and to stop requesting data, as the user is no longer using the app.

connection between the Pi and the Android phone, however, we could not utilize this connection within our Android app. There is not a large network of information available online on how to write code to send information from an Android app via Bluetooth or how to receive it on the Pi side. This made a Bluetooth connection difficult to work with. After a number of attempts, we decided to instead follow previous class projects and use TCP.

Although TCP connections require a router and for the internet to be accessible at all times, we found much more information online. As well, we saw past projects for this course using a TCP connection to send and receive information between Android and the Pi. First, we set up a connection using Python between the Pi and a laptop. This allowed us to experiment with example code and understand how socket programming works. At this point, we decided to switch from Native Script to Kotlin for our Android app. Since Kotlin is more similar to Java than Python, we set up a connection between a laptop and the Pi using a combination of Java and Python. The Pi was going to run on Python, so we kept that side of the connection in Python. On the laptop side, we moved from Python to Java. Finally, we were able to write a socket server in Kotlin.

Another design choice for socket programming was making the Pi a client and the app a server. Initially, we had the Pi as a server and the app as the client. However, a client polls information from the server. Instead of the app pushing information to the Pi, we wanted the Pi to request information from the app.

This allowed for simpler code on both sides. Additionally, we made the Pi sit and wait for a connection, instead of looking and immediately closing out. We felt that this was more natural for the user as the user could begin sending information whenever they wanted.

IMU Magnetometer Angle

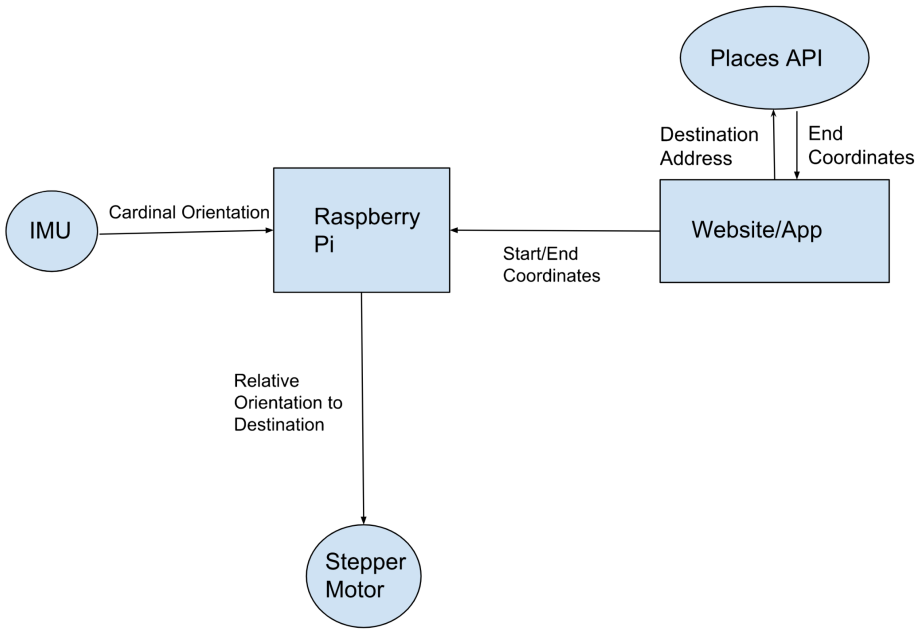
A magnetometer sends acceleration, gyro, and magnetic information to the Pi. For our design, we needed the magnetic information from the magnetometer. Since the Earth has its own magnetic field, the magnetic information allows us to calculate the angle relative to polar North. Using trigonometry, we were able to calculate this angle from the given data. This gave us our “local” angle.

Global Bearing Calculation

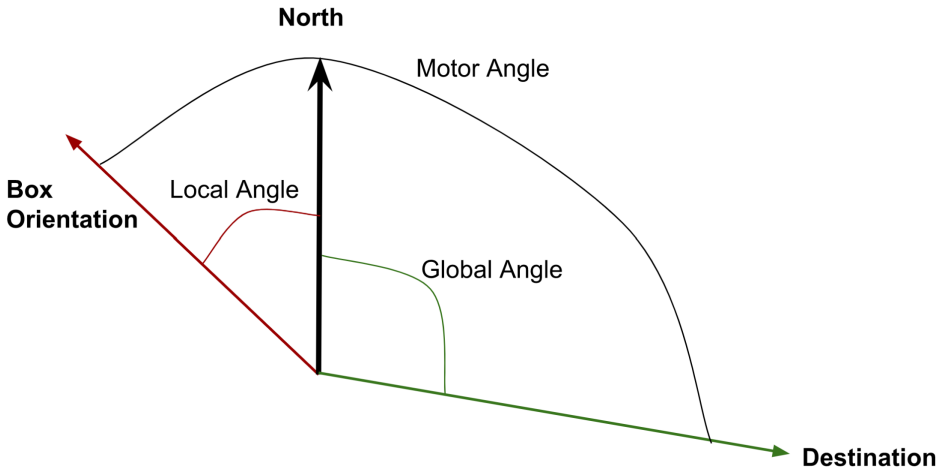
Next, we needed to determine the “global” angle of the compass. Initially, we used the longitude and latitude coordinates with numpy.arctan2, although this gave inconsistent results. After some research, we learned about bearing angles. Since the Earth is not flat, we must take its curvature into account when determining the angle between two coordinates. To account for this, we found a library called Geodesic that allowed us to easily calculate the bearing or “global angle” of travel which goes towards our destination.

Drawings

Module Overview



Angle Calculation



Testing

The main aspect of our project that required rigorous testing was combining our local and global angles. Since this was the summation of our project, any bugs at this point revealed themselves. Upon full assembly, we sent our compass a location North, South, East, and West of our position. We then compared the dial's angle to Google Maps. After some tweaking, we were able to successfully point to North and East locations. However, we found our compass struggling greatly with pointing South. It became apparent that we had issues with our math. Our compass movement and speed worked well. Additionally, we could turn our compass body while it pointed toward a Northern destination and watch it compensate for its new local orientation.

After reviewing our math, we decided to switch to a library for bearing angles described above. We selected this library due to its information available online and ease of use. Since it was the final step of our project, we spent many lab sessions checking and fixing our angles. We added many print statements to display the current coordinates, destination coordinates, local angle, and global angle. Since we found ourselves going back and forth between different trigonometric representations of bearing or global angles, we opted for a library. Since there was online evidence that the library gave consistent and accurate results, we decided to utilize it.

Over the course of this project, we spend multiple lab sections testing our angles. This was the most noticeable aspect of the project to test as other portions either worked or they did not.

Result

Our magic compass was successful. Our objective was to design a compass that points in the direction of a user-designated destination. We were able to successfully read user input text, utilize a web API, transmit current and destination location information, read direction from a magnetometer, process this data, and see the physical results with a stepper motor to control a compass dial.

One major change we made was switching from Bluetooth to TCP connection. We were unable to send and receive information over a Bluetooth connection from the app to the Pi. Although Bluetooth took time

to debug and work on, we learned about app coding and were able to better flush out our final design. It was important for us to understand our own priorities and work through a large obstacle. Despite the Bluetooth hiccup, we were able to accomplish our final goal and work through challenges. The final design was compact and efficient.

If we were to do this project again, we would have switched the coding language we were writing in earlier. Since only one partner had any experience with app development, we stuck with what we knew. Writing an app is daunting and takes time to learn about. As well, there is only so much information available online for each specific project type, making it challenging to branch out.

As a final project, the magic compass was an appropriate engineering design problem. Bringing together hardware and software components to create a functional device was rewarding. We were able to come to the lab with an idea and prior knowledge from the course projects and walk out with a working prototype. As well, the project kept the spirit of the course and aligned with past work to further our education in the intersection of hardware and software project design.

Future Ideas:

While we were satisfied with the outcome of our project, there are several features that we would have liked to add with more time. First, the cardinal orientation of the product is currently determined purely with a magnetometer. This worked better than we expected, but it is still noisy and susceptible to magnetic interference from nearby electronics and power sources. The magnetometer we are using is part of a 9 DoF IMU which also includes a gyroscope and accelerometer. If we applied a sensor fusion technique such as a Kalman filter to these three sensing modules then we could obtain a more accurate and stable measurement of orientation. Second, there are several ways we could expand how the user enters their desired destination. For example, we could attach a microphone to the Raspberry Pi module and employ text-to-speech technology to obtain a destination, effectively cutting out the smartphone. Another option would be attaching a map to the app and allowing users to tap on locations to select them as destinations. Third, it might be advantageous to the user to add a display to the product that presents the distance remaining between the person and their destination. To maintain the screen-free aesthetic of the product the display would need to be implemented as a set of dials.

Work Distribution

Project group picture

Elizabeth

emg229@cornell.edu

Oversaw TCP exchange, magnetometer interfacing, and circuitry design. Also took the lead on managing the project and fulfilling project update reports.

Joey

jah569@cornell.edu

Focused on Android development, stepper motor control, and fabricating the physical box. Also drove ideation of the product and its desired behavior.

Parts List

- Raspberry Pi 4 - $35.00
- Android Note 6 Phone - already possessed by Joey, price unknown
- 28BYJ-48 DC 5V Stepper Step Motor + ULN2003 Driver - $6.99
- Adafruit LSM6DSOX + LIS3MDL (https://www.adafruit.com/product/4517) - $19.95
- Clear acrylic box with custom raster - Laser cut from CMC scrap material
- Protoboard, tape and wires - Provided in lab

Total: $61.94

References

Android/Google Places Guide (https://developers.google.com/codelabs/maps-platform/places-101-android-kotlin)

Stepper Motor Guide (https://github.com/gavinlyonsrepo/RpiMotorLib/blob/master/Documentation/28BYJ.md)

Magnetometer Library (https://learn.adafruit.com/lis3mdl-triple-axis-magnetometer/python-

circuitpython)Geographic Resource (https://geographiclib.sourceforge.io/html/python/code.html)

Code Appendix

Raspberry Pi Code:

```
# client side
import socket
import time
import board
from adafruit_lsm6ds.lsm6dsox import LSM6DSOX as LSM6DS
import busio
import math
from adafruit_lis3mdl import LIS3MDL
import RPi.GPIO as GPIO
from RpiMotorLib import RpiMotorLib
import numpy as np
from geographiclib.geodesic import Geodesic

## TCP CONNECTION
HOST = "172.20.10.7" # PHONE IP
PORT = 65432
just_waited = False
connected = False
running = True
last_e = ""

## MAGNETOMETER
# ~ i2c = board.I2C()  # uses board.SCL and board.SDA
i2c = busio.I2C(board.SCL, board.SDA)
print("1")
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a microcontroller
accel_gyro = LSM6DS(i2c, address =0x6a)
print("2")
time.sleep(1)
mag = LIS3MDL(i2c, address =0x1c)
print("3")

## MOTOR
GpioPins = [17, 18, 27, 22]
mymotortest = RpiMotorLib.BYJMotor("MyMotorOne", "28BYJ")
```

```
# angle from top of motor out of 360
motor_angle = 0

def move(end):
        global motor_angle
        diff = (end - motor_angle) % 360
        motor_angle = end
        if diff > 180:
                diff -= 360
        coord = int(diff*512/360)
        if coord == 0:
                return
        elif coord < 0:
                mymotortest.motor_run(GpioPins , 0.005, -coord, True, False, "half", .05)
        else:
                mymotortest.motor_run(GpioPins , 0.005, coord, False, False, "half", .05)

def calcGlobalAngle(curr_lat, curr_long, dest_lat, dest_long):
        distLongitude = (dest_long - curr_long)

        # ~ y = math.sin(distLongitude) * math.cos(dest_lat)
        # ~ x = math.cos(curr_lat) * math.sin(dest_lat) - math.sin(curr_lat) * math.cos(dest_lat) * math
        # ~ bearing = math.atan2(y, x) * 180 / math.pi

        # ~ x = np.cos(np.deg2rad(dest_lat)) * np.sin(np.deg2rad(distLongitude))
        # ~ y = np.cos(np.deg2rad(curr_lat))* np.sin(np.deg2rad(dest_lat)) - np.sin(np.deg2rad(curr_lat))
        # ~ bearing = np.arctan2(x, y)
        # ~ bearing = np.degrees(bearing+225) % 360

        # ~ bearing = np.degrees(np.arctan2(dest_lat - curr_lat, dest_long - curr_long)) - 90
        # ~ bearing = -bearing

        bearing = Geodesic.WGS84.Inverse(curr_lat, curr_long, dest_lat, dest_long)['azi1']
        return bearing


with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        while(running):
                try:
                        s.connect((HOST, PORT))
                        connected = True
                except Exception as e:
                        print(e)
                        time.sleep(1)
```

```
 80                          move(0)
 81                          # ~ print("cant connect")
 82                          connected = False
 83                          # ~ move(0)
 84                  # ~ while(s.connect_ex((HOST, PORT)) != 0):
 85                          # ~ print("trying to connect")
 86                          # ~ time.sleep(1)
 87
 88                  # ~ s.sendall(b"Hello World Client Sent")
 89                  # ~ data = s.recv(1024)
 90
 91                  while(connected):
 92
 93                          try:
 94                                  ## TCP CONNECTIONH
 95                                  print("45")
 96                                  s.sendall(b"Hello World Client Sent")
 97                                  data = s.recv(1024)
 98                                  print(f"Received {data!r}")
 99                                  curr_dest = data.decode('utf-8').split(",")
100                                  if(curr_dest[0] == 'quit'):
101                                          move(0)
102                                          time.sleep(5)
103                                          s.close()
104                                          running = False
105                                          break
106                                  print(curr_dest)
107                                  curr_lat, curr_long, dest_lat, dest_long = float(curr_dest[0]), float(cu
108                                  print(curr_lat)
109                                  print(curr_long)
110                                  print(dest_lat)
111                                  print(dest_long)
112
113                                  ## MAGNETOMETER SECTION
114                                  acceleration = accel_gyro.acceleration
115                                  print("4")
116                                  gyro = accel_gyro.gyro
117                                  print("5")
118                                  magnetic = mag.magnetic
119                                  # ~ print(
120                                          # ~ "Acceleration: X:{0:7.2f}, Y:{1:7.2f}, Z:{2:7.2f} m/s^2".forr
121                                  # ~ )
122                                  # ~ print("Gyro          X:{0:7.2f}, Y:{1:7.2f}, Z:{2:7.2f} rad/s".format
123                                  # ~ print("Magnetic      X:{0:7.2f}, Y:{1:7.2f}, Z:{2:7.2f} uT".format(*r
```

```
                                # ~ print("")
                                print(f"magnetic[0]: {magnetic[0]} magnetic[1]: {magnetic[1]} magnetic[2
                                local_angle = math.atan2(magnetic[1], magnetic[0]) * 180 / math.pi
                                print(f"Local Angle from North: {local_angle}")

                                ## MOTOR
                                # ~ global_angle = math.atan2(dest_lat - curr_lat, dest_long - curr_long)
                                global_angle = calcGlobalAngle(curr_lat, curr_long, dest_lat, dest_long)
                                print(f"Global Angle from North: {global_angle}")
                                move(global_angle - local_angle)
                                print(f"Moving to {motor_angle}")

                                ## MORE TCP STUFF
                                just_waited = False
                                time.sleep(1)


                        except Exception as e:
                                move(0)
                                time.sleep(1)
                                print(e)
                                if(not just_waited):
                                        print("waiting for info")
                                        time.sleep(1)
                                just_waited = True
                                s.close()
                                connected = False

move(0)
```

ew raw
ttps://gist.github.com/joeyhz/943ad14518c47582515c8c26735dcff7/raw/7207a5583c1b389ca2127027aad09f7433b0ba21/final_code.py)
final_code.py (https://gist.github.com/joeyhz/943ad14518c47582515c8c26735dcff7#file-final_code-py) hosted with ❤ by GitHub (https://github.com)

Android Code:

```
/*
 * Adapted from the official Google Places start-up developer guide:
 * https://developers.google.com/codelabs/maps-platform/places-101-android-kotlin
 * which uses the Apache 2.0 license:
 * http://www.apache.org/licenses/LICENSE-2.0
 */

```

```
 8  package com.google.codelabs.maps.placesdemo
 9
10  import android.Manifest
11  import android.annotation.SuppressLint
12  import android.content.pm.PackageManager
13  import android.os.Bundle
14  import android.os.Handler
15  import android.os.Looper
16  import android.text.method.ScrollingMovementMethod
17  import android.util.Log
18  import android.widget.Button
19  import android.widget.TextView
20  import android.widget.Toast
21  import androidx.annotation.RequiresPermission
22  import androidx.appcompat.app.AppCompatActivity
23  import androidx.core.app.ActivityCompat
24  import androidx.core.content.ContextCompat
25  import androidx.lifecycle.lifecycleScope
26  import com.google.android.libraries.places.api.Places
27  import com.google.android.libraries.places.api.model.Place
28  import com.google.android.libraries.places.api.model.PlaceLikelihood
29  import com.google.android.libraries.places.api.net.FindCurrentPlaceRequest
30  import com.google.android.libraries.places.api.net.FindCurrentPlaceResponse
31  import com.google.android.libraries.places.api.net.PlacesClient
32  import com.google.android.libraries.places.ktx.api.net.awaitFindCurrentPlace
33  import com.google.android.libraries.places.ktx.widget.PlaceSelectionError
34  import com.google.android.libraries.places.ktx.widget.PlaceSelectionSuccess
35  import com.google.android.libraries.places.ktx.widget.placeSelectionEvents
36  import com.google.android.libraries.places.widget.AutocompleteSupportFragment
37  import kotlinx.coroutines.ExperimentalCoroutinesApi
38  import kotlinx.coroutines.launch
39  import java.io.*
40  import java.net.InetSocketAddress
41  import java.net.ServerSocket
42  import java.net.Socket
43  import java.util.concurrent.Executors
44
45  @Volatile private lateinit var destText : String
46  @Volatile private lateinit var currText : String
47  private var quitting = false
48
49  @ExperimentalCoroutinesApi
50  class AutocompleteActivity : AppCompatActivity() {
51      private  lateinit var responseView: TextView
```

```
52      private lateinit var responseViewCurrent: TextView
53      private lateinit var placesClient: PlacesClient
54      private lateinit var currentButton: Button
55      private lateinit var connectButton: Button
56      private lateinit var quitButton: Button
57
58      override fun onCreate(savedInstanceState: Bundle?) {
59          super.onCreate(savedInstanceState)
60          setContentView(R.layout.activity_autocomplete)
61
62          // Set up view objects
63          responseView = findViewById(R.id.autocomplete_response_content)
64          val autocompleteFragment =
65              supportFragmentManager.findFragmentById(R.id.autocomplete_fragment)
66                      as AutocompleteSupportFragment
67
68          // Specify the types of place data to return.
69          autocompleteFragment.setPlaceFields(listOf(Place.Field.NAME, Place.Field.ID, Place.Field.LAT_LNG
70
71          // Listen to place selection events
72          lifecycleScope.launchWhenCreated {
73              autocompleteFragment.placeSelectionEvents().collect { event ->
74                  when (event) {
75                      is PlaceSelectionSuccess -> {
76                          val place = event.place
77                          destText = "" + place.latLng.latitude + "," + place.latLng.longitude
78                          responseView.text = place.name + "(" + destText + ")"
79                      }
80                      is PlaceSelectionError -> Toast.makeText(
81                          this@AutocompleteActivity,
82                          "Failed to get place '${event.status.statusMessage}'",
83                          Toast.LENGTH_SHORT
84                      ).show()
85                  }
86              }
87          }
88
89
90          ///////////CURRENT PLACE//////////////////
91          // Retrieve a PlacesClient (previously initialized - see DemoApplication)
92          placesClient = Places.createClient(this)
93
94          // Set view objects
95          connectButton = findViewById(R.id.connect_button)
```

```
96          quitButton = findViewById(R.id.quit_button)
97          responseViewCurrent = findViewById(R.id.current_response_content)
98
99
100
101         quitButton.setOnClickListener {
102             quitting = true
103         }
104
105         // Set listener for initiating Current Place
106         val mainHandler = Handler(Looper.getMainLooper())
107         mainHandler.post(object : Runnable {
108             override fun run() {
109                 if (!quitting)
110                     checkPermissionThenFindCurrentPlace()
111                 mainHandler.postDelayed(this, 1000)
112             }
113         })
114
115         // Set listener for TCP
116         connectButton.setOnClickListener {
117             connectDevice()
118         }
119     }
120
121     /***
122      * TCP server which reports current and destination location when contacted
123      */
124     private fun connectDevice() {
125         connectButton.isEnabled = false
126         connectButton.isClickable = false
127         val server = ServerClass()
128         server.start()
129     }
130
131     class ServerClass() : Thread() {
132
133         lateinit var serverSocket: ServerSocket
134         lateinit var inputStream: InputStream
135         lateinit var outputStream: OutputStream
136         lateinit var socket: Socket
137
138         override fun run() {
139             try {
```

```
140                 serverSocket = ServerSocket(65432)
141                 socket = serverSocket.accept()
142                 inputStream = socket.getInputStream()
143                 outputStream = socket.getOutputStream()
144             } catch (ex: IOException) {
145                 ex.printStackTrace()
146             }
147 
148             val executors = Executors.newSingleThreadExecutor()
149             val handler = Handler(Looper.getMainLooper())
150             executors.execute(Runnable {
151                 kotlin.run {
152                     val buffer = ByteArray(1024)
153                     var byte: Int
154                     while (true) {
155                         try {
156                             byte = inputStream.read(buffer)
157                             if (byte > 0) {
158                                 var finalByte = byte
159                                 handler.post(Runnable {
160                                     kotlin.run {
161                                         var tmpMessage = String(buffer, 0, finalByte)
162                                         Log.i("Server class", "$tmpMessage")
163                                     }
164                                 })
165                                 // Send Location Info:
166                                 if (quitting) write(("quit").toByteArray())
167                                 else {
168                                     Log.i("Sending", "Sending: $currText,$destText")
169                                     write(("$currText,$destText").toByteArray())
170                                 }
171 
172                             }
173 
174                         } catch (ex: IOException) {
175                             ex.printStackTrace()
176                         }
177                     }
178                 }
179             })
180         }
181 
182         private fun write(byteArray: ByteArray) {
183             try {
```

```
                Log.i("Server write", "$byteArray sending")
                outputStream.write(byteArray)
            } catch (ex: IOException) {
                ex.printStackTrace()
            }
        }
    }


    /**
     * Checks that the user has granted permission for fine or coarse location.
     * If granted, finds current Place.
     * If not yet granted, launches the permission request.
     */
    private fun checkPermissionThenFindCurrentPlace() {
        when {
            (ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.ACCESS_FINE_LOCATION
            ) == PackageManager.PERMISSION_GRANTED || ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.ACCESS_COARSE_LOCATION
            ) == PackageManager.PERMISSION_GRANTED) -> {
                // You can use the API that requires the permission.
                findCurrentPlace()
            }
            shouldShowRequestPermissionRationale(Manifest.permission.ACCESS_FINE_LOCATION)
            -> {
                Log.d(TAG, "Showing permission rationale dialog")
            }
            else -> {
                // Ask for both the ACCESS_FINE_LOCATION and ACCESS_COARSE_LOCATION permissions.
                ActivityCompat.requestPermissions(
                    this,
                    arrayOf(
                        Manifest.permission.ACCESS_FINE_LOCATION,
                        Manifest.permission.ACCESS_COARSE_LOCATION
                    ),
                    PERMISSION_REQUEST_CODE
                )
            }
        }
    }

```

```
228       @SuppressLint("MissingPermission")
229       override fun onRequestPermissionsResult(
230           requestCode: Int,
231           permissions: Array<String>, grantResults: IntArray
232       ) {
233           if (requestCode != PERMISSION_REQUEST_CODE) {
234               super.onRequestPermissionsResult(
235                   requestCode,
236                   permissions,
237                   grantResults
238               )
239               return
240           } else if (permissions.toList().zip(grantResults.toList())
241                   .firstOrNull { (permission, grantResult) ->
242                       grantResult == PackageManager.PERMISSION_GRANTED && (permission == Manifest.permissio
243                   } != null
244           )
245           // At least one location permission has been granted, so proceed with Find Current Place
246               findCurrentPlace()
247       }
248
249       /**
250        * Fetches a list of [PlaceLikelihood] instances that represent the Places the user is
251        * most likely to be at currently. Then takes the longitude and latitude of most likely place
252        * and saves them in currText
253        */
254       @RequiresPermission(anyOf = [Manifest.permission.ACCESS_COARSE_LOCATION, Manifest.permission.ACCESS_
255       private fun findCurrentPlace() {
256           // Use fields to define the data types to return.
257           val placeFields: List<Place.Field> =
258               listOf(Place.Field.LAT_LNG)
259
260           // Use the builder to create a FindCurrentPlaceRequest.
261           val request: FindCurrentPlaceRequest = FindCurrentPlaceRequest.newInstance(placeFields)
262
263           // Call findCurrentPlace and handle the response (first check that the user has granted permissio
264           if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) ==
265               PackageManager.PERMISSION_GRANTED ||
266               ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION) ==
267               PackageManager.PERMISSION_GRANTED
268           ) {
269               // Retrieve likely places based on the device's current location
270               lifecycleScope.launch {
271                   try {
```

```
272                     val response = placesClient.awaitFindCurrentPlace(placeFields)
273                     currText = response.prettyPrint()
274                     responseViewCurrent.text = currText
275
276                     // Enable scrolling on the long list of likely places
277                     val movementMethod = ScrollingMovementMethod()
278                     responseView.movementMethod = movementMethod
279                 } catch (e: Exception) {
280                     e.printStackTrace()
281                     responseView.text = e.message
282                 }
283             }
284         } else {
285             Log.d(TAG, "LOCATION permission not granted")
286             checkPermissionThenFindCurrentPlace()
287
288         }
289     }
290
291     companion object {
292         private val TAG = "CurrentPlaceActivity"
293         private const val PERMISSION_REQUEST_CODE = 9
294     }
295 }
296
297 fun FindCurrentPlaceResponse.prettyPrint(): String {
298     return this.placeLikelihoods[0].place.latLng.latitude.toString() + ","+
299             this.placeLikelihoods[0].place.latLng.longitude.toString()
300 }
```

view raw (https://gist.github.com/joeyhz/eb045539dcf6b4e2fa3c02590fe83601/raw/87c3ccd837ade27eedb9dbdc082c7abec722bef8/Main.kt) Main.kt (https://gist.github.com/joeyhz/eb045539dcf6b4e2fa3c02590fe83601#file-main-kt) hosted with ❤ by **GitHub (https://github.com)**

Code is displayed through GitHub gists so it will not appear without an Internet connection.