**Raspberry Pi 4 Navigation System for the Visually Impaired**

**A Thesis**

**Presented to**

**Dr. Sridhar Ramachandran**

**and the Informatics Department at**

**Indiana University Southeast**

**In Partial Fulfilment of The Requirements for a Minor in Informatics**

**By David Conrad**

**January 2020**

**Abstract**

**Visually impaired people face numerous challenges in daily life. Foot navigation has been greatly improved for all people because of innovations by companies like Google and Apple. Their navigation applications are easily used by those with visual impairments. However, these applications fail to adequately map Indiana University’s Campus. To solve this problem, this investigation created a system that allows the visually impaired to independently navigate Indiana University’s campus using a raspberry pi and custom mapping software developed in Python. The self-contained system can be held with one hand or in a bag and only requires a charged battery and a GPS satellite connection to function. The system directs the user via headphones to their destination by issuing voice turn-direction and distance. The developed system, in coordination with a guide dog or a walking stick, allows the user to find any of the pre-programmed locations without sight, safely along walking paths.**

**Background**

The main navigational aids of Indiana University’s campus in 2020 is signage and walkways that direct students and visitors towards the correct buildings. The visually impaired, with the help of walking canes and service dogs, navigate via memory and landmarks. At the beginning of classes, many students use signage to find buildings for their classes. The visually impaired, without aid, are not able to read the signage. Indiana University Southeast is not mapped in depth by Google Maps or other services. According to Glenk (2019), service dogs greatly improve the quality of life and independency of the blind. Guide dogs are not able to help navigate in foreign territory as they or the owner does not have the route memorized. This is a significant detriment to the blind’s independence who would need a person to guide them around campus the first few days of classes.

**Question**

How can the raspberry pi be used to help the visually impaired navigate outdoors on the IUS campus? The following sub questions will also need to be answered. Is a USB GPS device accurate enough for the project? Is there a viable speech synthesis tool usable within a python program? Is there a viable speech to text tool usable within the python and raspberry pi ecosystem?

**Hypothesis**

The python program will be able to run in the Raspbian environment, pull GPS data from the usb GPS puck, initiate on start-up without a monitor, record and interpret speech from the user, and narrate information back to the user. The Raspberry Pi 4 will be able to draw enough power from a portable battery to stably run. Finally, given the rest is true, the python program will be able to use the location data from the GPS data to determine the direction and distance the user is from the desired location and guide the user using speech synthesis.

**Flow**

The flow of the project was as follows:

1. Attain and test a Raspberry Pi, GPS USB puck, portable battery, usb numpad, and headphones.
2. Install required libraries and daemon tools and write program in python.
3. Test at my house and at IUS

**Supporting Research**

Raspberry Pi

The Raspberry Pi is a small, inexpensive, single board computer. Originally launched in 2012, the Pi has gone through numerous iterations to stay up to date with the newest interface options and computing power. The current iteration is the Raspberry Pi 4. It includes two micro-HDMI ports, two USB 2.0, two USB 3.0, ethernet, Bluetooth, wireless internet, a micro-SD card slot and a 3.5mm audio jack. It is powered via USB-c power. The Pi 4 comes with a Broadcom BCM2711 quad core 64 bit 1.5 GHZ processor. The Raspberry Pi was originally developed as a cheap educational device to introduce children to programming. The pi is used commonly in-home automation and project prototyping. This information was attained from Raspberry Pi’s website, raspberrypi.org.

Global Positioning System

Navigation has been paramount to the trade and globalization throughout human history. Some of the earliest civilizations grew as a direct result of finding new ways to navigate. Guidance by the stars and a compass was the gold standard for much of human history. In the late 1970s, a new kind of navigation began development by the United States military. According to Tsui (2000), the GPS or Global Positioning Satellites had 24 satellites in orbit by the end of 1993. Si and Aung (2011) report that 24 satellites are required to accurately locate the receiver. Generally, GPS receivers have 9 feet in precision, but the puck I am using gives a precision measurement. The satellites measure the locations of the satellites, which then the software interprets and calculate its location via distance between it and the satellites and the speed of light.

NMEA Data

According to Si and Aung (2011), NMEA stands for National Marine Electronics Association. NMEA is the defined format for GPS receiver data given to computers. The data includes position, velocity, time, number of satellites and a few other data points. The format is a compact way of giving information to a computer. It is given as a string of numbers and commas. The numbers are in order and the commas separate the numbers. The order determines the data point. NMEA sentences are translated by GPSD.

GPSD

GPSD is a service daemon, or a program that runs in the background of a computer. For this project, I have set GPSD to run on boot. GPSD takes the NMEA data provided by a GPS receiver through a serial or USB port and translates it to plain English. The daemon provides several functions to access the data in various programming languages. The supported languages are perl, python, C++, C, and Java. While GPSD runs with zero configuration on Linux, I found it needs a few fixes to run properly in Raspbian Buster.

Festival Text to Speech

Festival text to speech or flite is an open source text to speech engine. According to the information on the flite GitHub repository (2017), flite is compatible with various Linux systems. The library was developed by Alan Black of Carnegie Melon. After installing the libraries, they can be accessed in python by using the system command to directly enter into the command line.

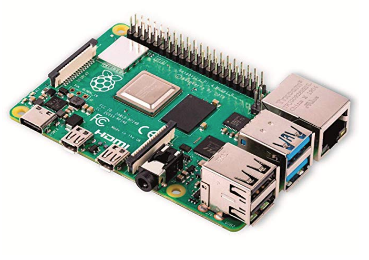
Python

Python is a popular programming language used for in a variety of different ways. It’s simplicity to eye and vast power have allowed it to rise a one of the most used programming languages today. According to the Python Software Foundation (2020), Python can be used for Web and Internet development, for scientific and numeric computing, to develop desktop GUIS, and as a general software development language. Python is able to run without compiling, making it quick and easy to set up scripts to run in command line. I chose for this project, as I wanted to learn a new language and I knew python batch scripts could be easily set up to run on boot. Python’s popularity was a major factor in my choice. Because of the popularity, the community on stack exchange and other website made learning and implementing python very easy.

Raspbian Operating System

According to the Linux Foundation (2020), Linux is an open source operating system that provides a bootloader, kernel, init system, daemons, graphical server, desktop environment, and applications. Raspbian Buster is the latest release, as of February 2020, of the primary operating system for raspberry pi It is a modified Debian Wheezy Linux Distribution. Raspbian provides several text editors for easier programming language writing. Installing and upgrading Raspbian is easy with an SD card and a few terminal commands. Raspbian is maintained by the Raspberry Pi Foundation. (2020) Terminal Commands are shared with Debian and can be found at on raspberry pi’s website.

**Developed System**

 There is a hardware and software component to the system. The base of the hardware is a Raspberry Pi 4. With the help of software, a tactile numpad, 3.5mm headphones, USB GPS puck, and external mobile battery are all required for the system. The software utilizes Raspbian’s latest release, buster. The software is written in python and executed by the Raspberry Pi on boot.

Power Bank

Raspberry Pi 4

Tactile Numpad



USB GPS Puck

3.5mm Headphones

**Steps to Replicate**

In this Section, I outline the steps I took to create the system and how to replicate it.

**Required Materials:**

Raspberry Pi 4

3.5mm Speaker or Headphones

USB GPS Puck

Battery Bank Capable of Providing 3 Amps

Micro HDMI to HDMI

USB C to USB C Cable

Micro SD Card and Reader

Another Computer for Remote Desktop

Keyboard

Mouse

Monitor

**Step 1: On a Separate Computer:**

Format your SD card with SD Card Formatter.

Download NOOBS Raspberry Pi’s Website

Extract onto the micro SD card with SD card reader

**Step 2: Power Up and Install NOOBS on Raspberry Pi**

Insert the SD card

Plug in Micro HDMI to the PI and Monitor

Plug in Keyboard/Mouse

Plug in USB GPS Puck

Plug in Headphones in 3.5mm Jack

Plug power into wall/Battery Pack and Pi via USB C

Install Rasbian Full

Follow the set-up prompts

Connect to the Internet

Open Command Line

sudo apt-get update

sudo apt-get upgrade

Updates Raspbian OS

Sudo reboot now

**Step 3: Set Up Windows Remote Desktop**

Open Command Line

Sudo apt-get install xrdp

Installs xdrp

Hostname -I

Tells us the IP of the Raspi. Save it

Sudo reboot now

Reboots Raspberry Pi

Open Windows Remote Desktop on another Computer

Log in via the IP

Log in with default user and password

Pi (Lowercase)

Raspberry (lowercase)

**Step 4: Install Python**

In Command Line

Sudo apt-get install python 3

Sudo apt-get install python

**Step 5: Install and Test GPSD**

The information on gpsd was found on adafruit (2014).

Sudo apt-get install GPSd GPSd-clients python GPSd

Installs GPSd package

Every time we run GPSd after reboot, we need to reset it. I found a bash script online that fixes this problem during boot.

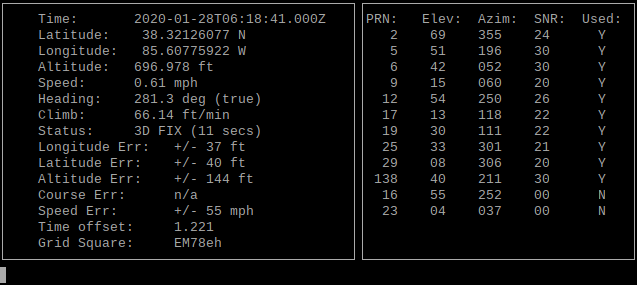
sudo systemctl stop GPSd.socket

Sudo systemctl disable GPSd.socket

Sudo killall GPSd

Sudo GPSd /dev/ttyUSB0 -F /var/run/GPSd.sock

We can run GPSd with cGPS -s



We should see something like this. If not, the likely issue is the USB GPS Puck is registering to a different port. We can find it by navigating to /dev/ folder and using the list command ls. Then, if we unplug it, list it, and replug, we should be able to find out what port it is in. Mine is “TTYUSB0”. Then, replace whatever it is with ttyUSB0.

**Step 6: Setting up the batch script so we never have to run those commands again to start GPSd.**

Laid out by Dexter Industries (2020). Put the lines at the end of the bash file via:

Sudo nano /home/pi/.bashrc

Opens .bashrc in nano

Add the Lines to the end (below fi)

sudo systemctl stop GPSd.socket

sudo systemctl disable GPSd.socket

sudo killall GPSd

sudo GPSd /dev/ttyUSB0 -F /var/run/GPSd.sock

**Step 7: Install and Test flite**

Install flite

sudo apt-get install flite-dev flite

Test

Flite -t “text here”

In python

Import os

os.system(‘flite -t “text here”’)

**Step 8: Install Necessary Python Libraries**

Geopy

Pip install geopy

**Step 9: GPS Poller Class**

Use the code provided by the Dan Mandle (2012) and the GPSD group to access the GPSD daemon tool. Edit the code, to remove everything but the try while, to continually access the nmea data, and the poller class, to access the data.

**Step 10: Import Python Libraries**

import geopy  
import os  
from GPS import \*  
from time import \*  
import time  
import threading  
import math

**Step 11: Using Festival Text to Speech to convey information to the user**

Using flite, propose the choices of location to the user. Get user choice from the user via the tactile keypad. Use the information to select the location inside the program. os.system sends a command directly to the terminal. According to Alan Black and Kevin Lenzo of Carnegie Melon (2001), Flite, short for Festival Lite, is a text to speech synthesis engine. After installing flite on the raspberry pi, I can directly access it via os.system. This allows me to communicate with the visually impaired user via headphones. I can relay all the information necessary to the user this way.

os.system('flite -t "Select From the List of Locations Using Your Keypad"')  
os.system('flite -t "Top Right Button: Life Sciences"')  
os.system('flite -t "Bottom Right Button: Admissions Office"')

**input1 = input()**

**if input1 == 7:**

**elif input1 == 1:**

**else:**

os.system('flite -t "not an option"')

**Step 12: Calculate Navigational Data**

Inside the if statement, I used the GPSD libraries to find the current latitude and longitude of the user via the usb GPS puck. Then, I calculate the angle from north of the line created by the user’s current latitude and longitude, and the desired location’s latitude and longitude. I used a tuple to store the latitude and longitude for organization purposes. I also used many math functions imported via math libraries, which come with python 3, to calculate the angle. The function GPSd.fix.latitude returns a decimal latitude for the current location of the USB GPS puck. GPSd.fix.longitude does the same for longitude. Next, I calculate the degrees from north, initial\_bearing.

P1 = [GPSd.fix.latitude, GPSd.fix.longitude]  
P2 = [38.32024131, -85.60533464]  
print  
'P1 ', P1  
print  
'P2 ', P2  
lat1 = math.radians(P1[0])  
lat2 = math.radians(P2[0])  
  
diffLong = math.radians(P1[1] - P2[1])  
  
x = math.sin(diffLong) \* math.cos(lat2)  
y = math.cos(lat1) \* math.sin(lat2) - (math.sin(lat1)  
 \* math.cos(lat2) \* math.cos(diffLong))  
  
initial\_bearing = math.atan2(x, y)

**Step 13: Calculate User Bearing and Use Navigational Data to Direct the User**

The GPSD program uses the NMEA data to calculate a track for the GPS puck. The track is an angle, degrees from north, of the USB Pucks path. As the user will keep the puck with them as they walk, it can quickly gain an accurate track of the user’s direction. Then, I compared the two tracks by subtracting them and created a series of if and else if statements. Each one will tell the user, using the same process as before, a direction to turn based on the comparison.

if GPSd.fix.track == 0:  
 os.system('flite -t "Zero heading"')  
elif -22.5 <= HD <= 22.5:  
 # os.system('flite -t "Negative 22.5 and 22.5"')  
 os.system('flite -t "Strait"')  
 print  
 'Between -22.5 and 22.5'  
  
elif 22.5 <= HD <= 67.5:  
 # os.system('flite -t "22.5 and 67.5"')  
 os.system('flite -t "slight left"')  
 print  
 'between 22.5 and 67.5'  
elif 67.5 <= HD <= 112.5:  
 # os.system('flite -t "67.5 and 112.5"')  
 os.system('flite -t "full right"')  
 print  
 'between 67.5 and 112.5'  
elif 112.5 <= HD <= 157.5:  
 # os.system('flite -t "112.5 and 157.5"')  
 os.system('flite -t "Big Right"')  
 print  
 'between 112.5 and 157.5'  
elif 157.5 <= HD <= 202.5:  
 # os.system('flite -t "157 and 202"')  
 os.system('flite -t "Turn Around"')  
 print  
 'between 157.5 and 202.5'  
elif 202.5 <= HD <= 247.5:  
 # os.system('flite -t "202.5 and 247.5"')  
 os.system('flite -t "Big left"')  
 print  
 'between 202.5 and 247.5'  
  
elif 247.5 <= HD <= 262.5:  
 # os.system('flite -t "247.5 and 262.5"')  
 os.system('flite -t "Full left"')  
 print  
 'between 247.5 and 262.5'  
elif 262.5 <= HD <= 337.5:  
 # os.system('flite -t "262.5 and 337.5"')  
 os.system('flite -t "Light Left"')  
 print  
 'between 262.5 and 337.5 Slight Right'  
elif 337.5 <= HD <= 360:  
 # os.system('flite -t "337.5 and 360"')  
 os.system('flite -t "Forward"')  
 print  
 'between 337.5 and 360. Forward'  
elif -67.5 <= HD <= -22.5:  
 os.system('flite -t "slight left"')  
elif -112.5 <= HD <= -67.5:  
 os.system('flite -t "full right"')  
elif -157.5 <= HD <= -112.5:  
 os.system('flite -t "Big Right"')  
elif -202.5 <= HD <= -157.5:  
 os.system('flite -t "Big left"')  
elif -247.5 <= HD <= -202.5:  
 os.system('flite -t "Big left"')  
elif -262.5 <= HD <= -247.5:  
 os.system('flite -t "Turn Around"')  
elif -337.5 <= HD <= -262.5:  
 os.system('flite -t "Light Left"')  
elif -360 <= HD <= -337.5:  
 os.system('flite -t "Forward"')  
  
else:  
 print   
 'HD Error'

**Step 14: Calculate Distance and Relay Distance to user**

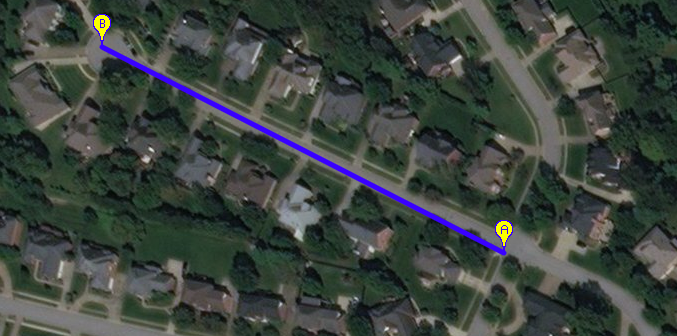
Finally, I use the geopy libraries to calculate the distance between the current user coordinates and the final location in feet.

d = geopy.distance.distance(P1,P2).ft

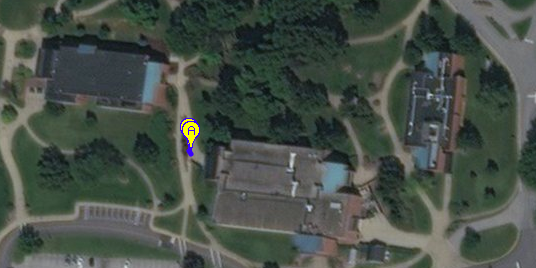
os.system(‘flite -t d’)

**Testing**

Testing of the project was done at my home in prospect Kentucky. Throughout the writing process, I used print statements in the command line to ensure the data was accurate and the calculations were correct. After, I had sufficiently decided everything was working, I replaced the GPS coordinates for IUS Life Sciences for coordinates down the street from me. The two coordinates and the ideal path between my starting location and the destination are shown on the map below. Then, I put the headphones in my ears, walked to location B, and listened to the feedback.



After I start walking, and the GPS can gain a track, the code looped and told me either slight right, slight left, left, right, large left, large right, turn around, or go straight. The device also says how many feet I am from the location. In the picture above, point b is about 700 feet from the location and that is what the device told me. It updates with new GPS data from the puck each loop, set for approximately 5 seconds. Following the voice-synthesized directions, I was able to navigate to my location. Within 30 feet of the coordinates, the program tells me I have arrived. Everything worked as intended. I replaced the coordinates shown with the ones for life sciences and admissions shown below.



Life Sciences: 38.34323, -85.81942



Admissions: 38.3447100,-85.8199700

**Conclusion**

To use the system, the user will hold the raspberry pi, GPS puck, battery, and mini-numpad in a small bag. The user will put the headphones in their ears and plug in the Raspberry Pi. Then, the system will introduce itself to help navigate to specific Indiana University Southeast buildings. In its current iteration, choices to navigate to the Life Sciences Building and the admissions office are available. The device will guide the user from either parking lot to the buildings, or between the buildings without obstacle. It would be simple, with only a few if statements, to add every building to the mapping system.

The Raspberry Pi combined with python and a few other components is capable of guiding a visually impaired user to various buildings on IUS’ campus. This does require the user to have a guide dog or walking stick in order to avoid obstacles and stay on sidewalks. However, the device would greatly improve the independence of the user, as they would not need to be led by another person to the various buildings on campus. This would be especially useful for a new, blind student who doesn’t have an establish internal map of the campus. They could use the device on their first few days of classes, until they had the campus memorized, to navigate to the various buildings where their classes are held.

**References**

Glenk et al. (2019). Perceptions on Health Benefits of Guide Dog Ownership in an Austrian

Population of Blind People with and without a Guide Dog. *Animals*. Retrieved from

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6680747/>

World Health Organization. (2019). Blindness and Vision Impairment. *WHO*. Retrieved

February 10, 2020.

<https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment>

Black and Lenzo. (2001) Festvox Speech Software. *Language Technologies Institute, Carnegie*

*Melon.* Accessed February 10, 2020 from <http://www.festvox.org/flite/> and

<http://www.cs.cmu.edu/~awb/papers/ISCA01/flite/flite.html>

(2020). Five Ways to Run a Program On Your Raspberry Pi At Startup. *Dexter Industries.*

Accessed January 20, 2020 from

<https://www.dexterindustries.com/howto/run-a-program-on-your-raspberry-pi-at-startup/>

Mandle. (2012). Python-gps Sample Codes. *Wolfg Github*. Accessed from

<https://gist.github.com/wolfg1969/4653340>

# (2014). Adafruit Ultimate GPS Hat for Raspberry Pi. *Adafruit.* Accessed from

# <https://learn.adafruit.com/adafruit-ultimate-gps-hat-for-raspberry-pi/use-gpsd>

Tsui, James. (2000). Fundamentals of Global Positioning System Receivers.

John Wiley and Sons, inc. Accessed from

<http://kakukoto.free.fr/F/019/Fundamentals%20of%20Global%20Positioning%20System%20Receivers/Fundamentals%20of%20Global%20Positioning%20System%20Receivers.pdf>

Si and Aung (2011). Position Data Acquisition from NMEA Protocol of Global Positioning

System. *International Journal of Computer and Electrical Engineering* Vol. 3, June 2011. Acquired from <http://www.ijcee.org/papers/340-E898.pdf>

GPSD – a GPS service daemon. Updated 2020. Unknown Original Date. Accessed from

<https://gpsd.gitlab.io/gpsd/index.html#documentation>

Black, Alan et al. (2017). Flite. *Github*. <https://github.com/festvox/flite>

(2020) Raspbian. *Raspberry Pi Foundation*. <https://www.raspberrypi.org/downloads/raspbian/>

(2020). What is Linux? *The Linux Foundation*. <https://www.linux.com/what-is-linux/>

(2020). Applications for Python. *Python Software Foundation*.

<https://www.python.org/about/apps/>

Albertson, Jeff. (2017). Explore Your New Campus with Google Maps. *Google.* Accessed from

<https://www.blog.google/products/maps/explore-your-new-campus-google-maps/#:~:text=>

(2020) Raspberry Pi 4. *Raspberry Pi Foundation.*

<https://www.raspberrypi.org/products/raspberry-pi-4-model-b>