CS 437
IOT LAB 1 Assignment Report
Self-Driving Car:
Design Considerations

- *by project Group Members:*

Full Name:	E-mail	NetID
Mr. Michael Potts	mjpotts2@illinois.edu	Mjpotts2
Mr.Pradeep Sakhamoori	ps44@illinois.edu	ps44

DEMO VIDEO URL:

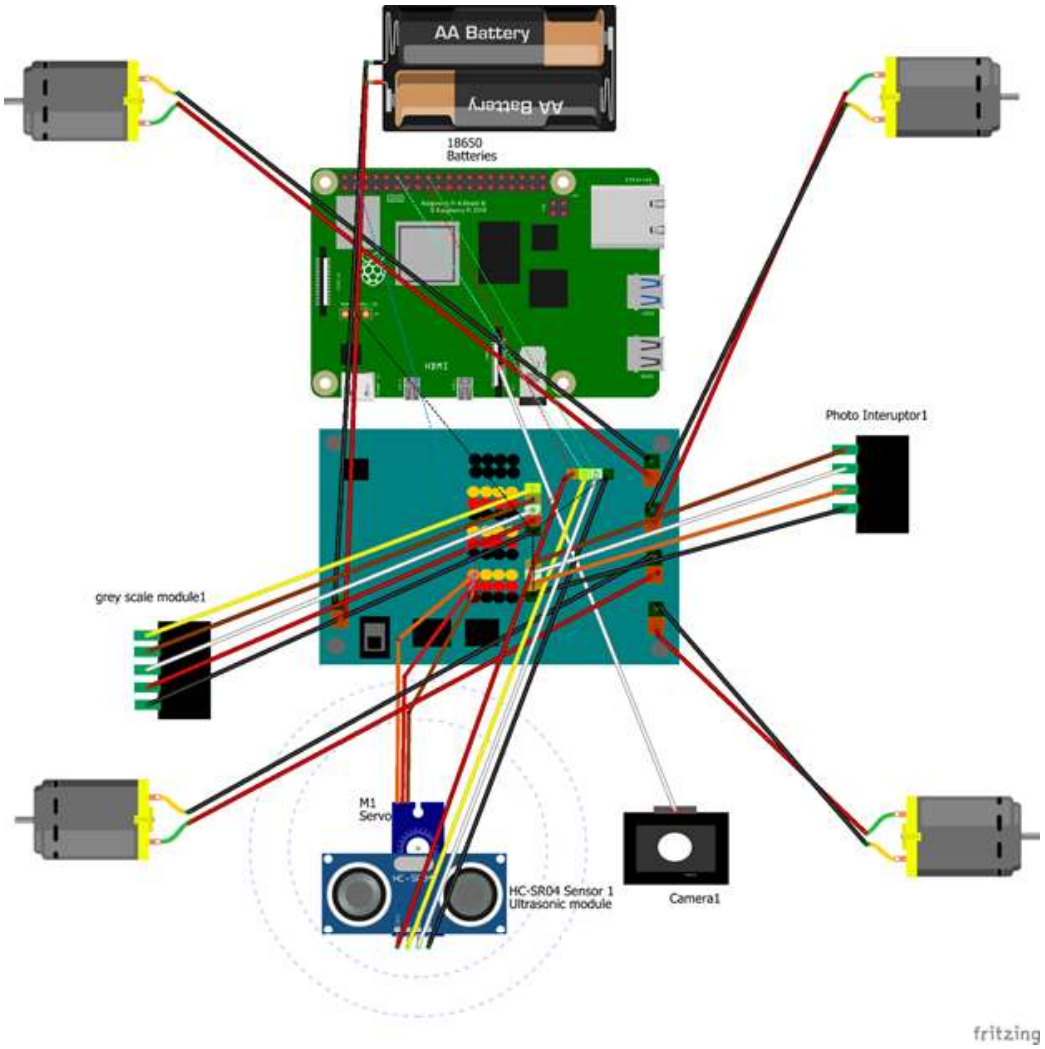
https://uofi.box.com/s/tmkygnivr1nl4anu9ezo2epm5819tlk8

Contents

1. IOT Lab 1: IoT Devices (Part 1)

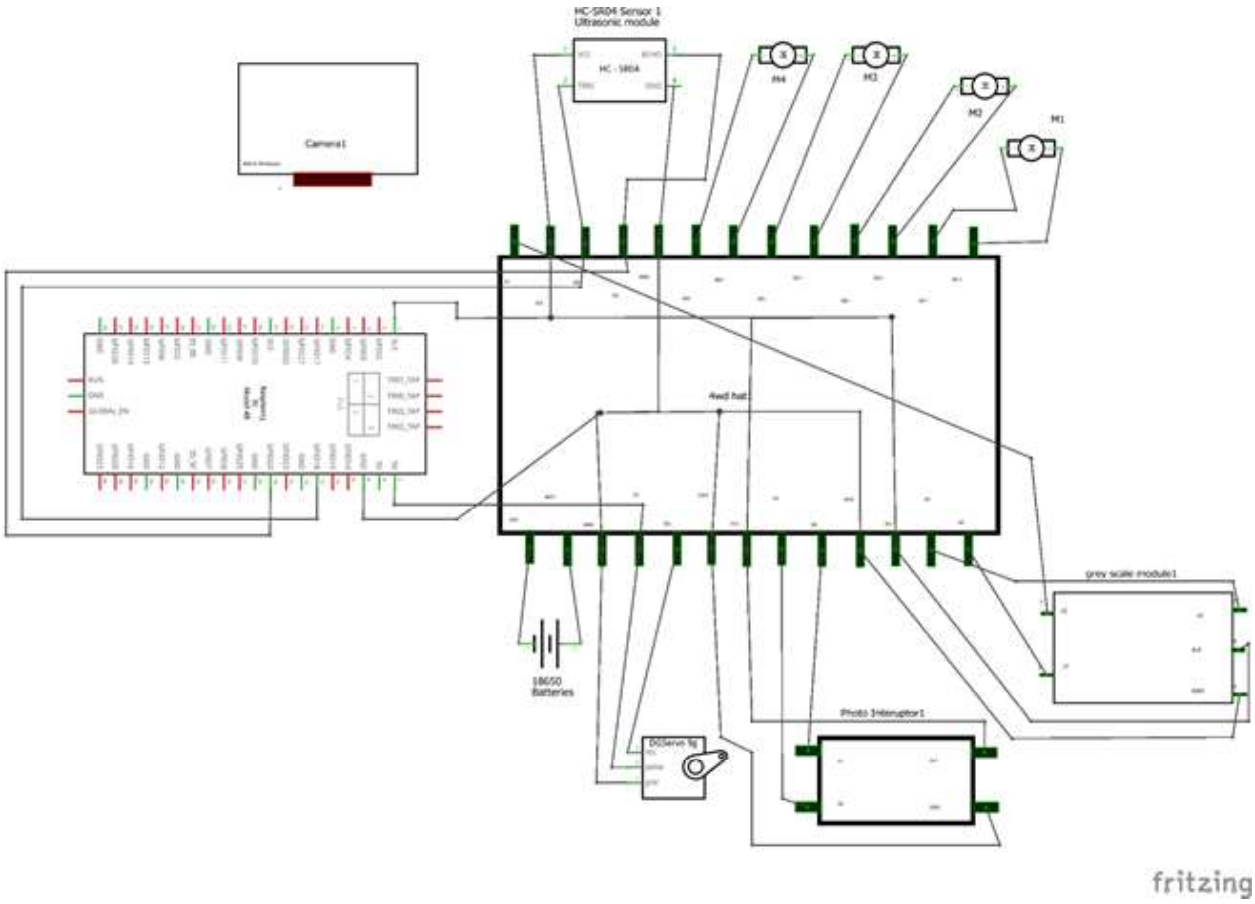
1.1. Topology

For the creation of the topological report, I attempted to find the relevant data sheets for each part, but was unfortunately unable to find a corresponding data sheet for both the grey scale module, and the 2-ch photo interrupter. Regardless, I was able to assume the proper connections for most all of the given components, and would assume that someone referring to our topology would understand how the hat module would sit upon the given pins for the Raspberry pi 4 board. To begin, the diagram overview in roughly the position where each component would sit upon the car:



Data sheet for the ultrasonic module:
https://tutorials-raspberrypi.com/raspberry-pi-ultrasonic-sensor-hc-sr04/

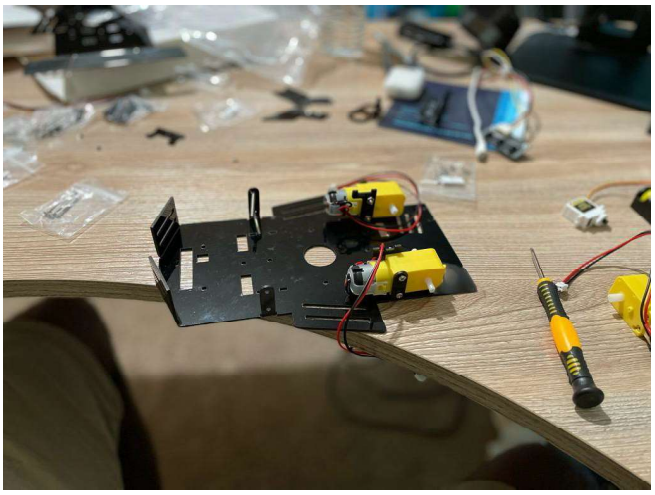
And secondly, the schematic diagram. Please zoom to view the labels. Several of the connected pins from the hat module to the Pi board were assumed (such as ground, 3.3v, 5v) and the pins for the ultrasonic were connected according to their data sheet. Due to the lack of data sheets for the other modules, the power pins were connected but the sensing pins were not connected to avoid introducing error. During the import process into fritzing, the provided text was shrunken and slightly skewed from its original position:

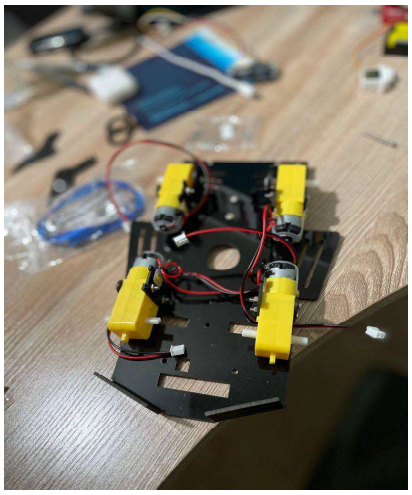


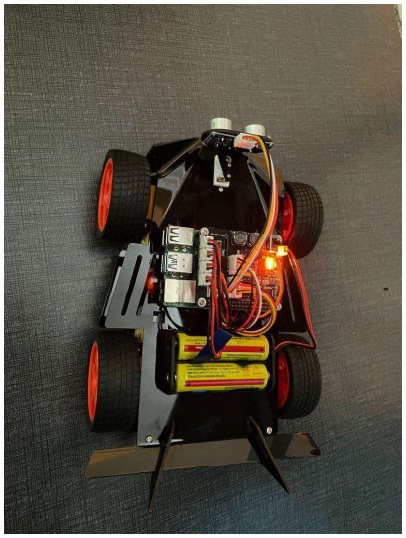
1.2. Car Assembly

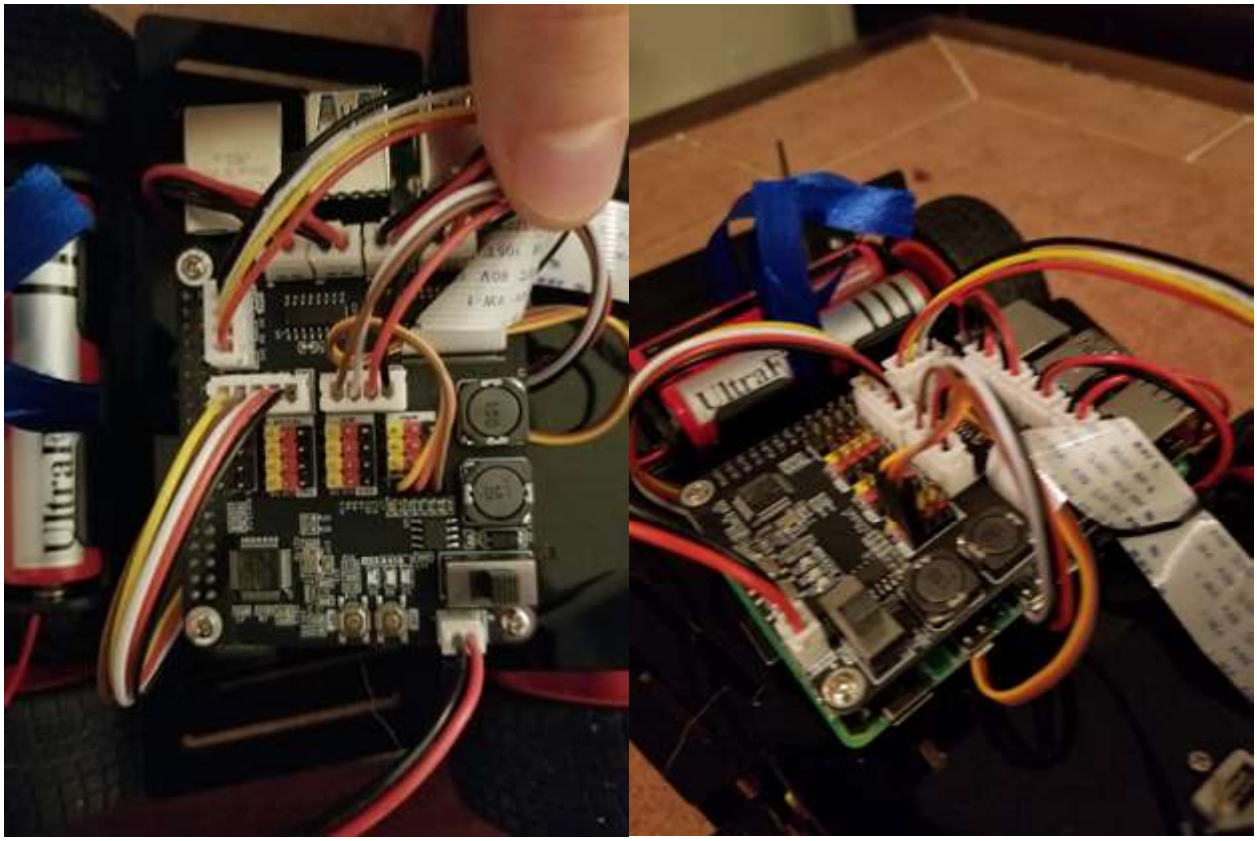
The car assembly, while straightforward, and quite enjoyable, was not without a few minor setbacks. The building instructions were clear and easy to understand. Unfortunately, one group member ran into difficulties while installing the Raspbian OS, receiving the error message: “end kernel panic – not syncing:Attempted to kill init! Exitcode=0x0000000b.” As the debugging process continued it was discovered that the instructions did not explicitly state to remove the microSD card from the USB-MicroSD reader. Once the SD card was inserted into the sd slot on the underside of the board, the problem was solved! Another issue a group member ran into, was with one of his components; specifically his ultrasonic module. He was able to order a replacement and the issue was resolved as well.


PREDATOR









1.3. Car moving

Through evaluating the given code and experimenting with the tests provided, it quickly became easy to alter the code to produce interesting new results. By scanning with the ultrasonic in varying ways, we realized that by changing the reference distance, it could become easy to extend the car's sight. Through re-writing the code, we could eventually gather more information beyond the status and distance, such as the reference angle, and step of the angle as it pans. From these initial steps, we were able to help the car make simple decisions about which direction to turn based on which side of the car displayed a clear status.

Leading into the next phase, we also discovered the sometimes unfortunate relationship between the level of speed and the required time to scan. By moving too quickly, it became obvious the car was unable to keep scanning in a timely manner, leading to crashes. This was easily fixed by finding an appropriate balance between the speed and scan time (somewhere roughly 30-35cm in distance and 25-28 speed).

1.4. Obstacle Avoidance

Finally our car was ready for our obstacle course. By balancing the speed and scan time, the car was able to navigate the obstacle course ⅘ times without incident, however, due to the holes in the cardboard walls, the car would occasionally miss and hit. Once these holes were patched, our vehicles' problems were resolved.

2. IOT Lab 1: IoT Devices (Part 2)

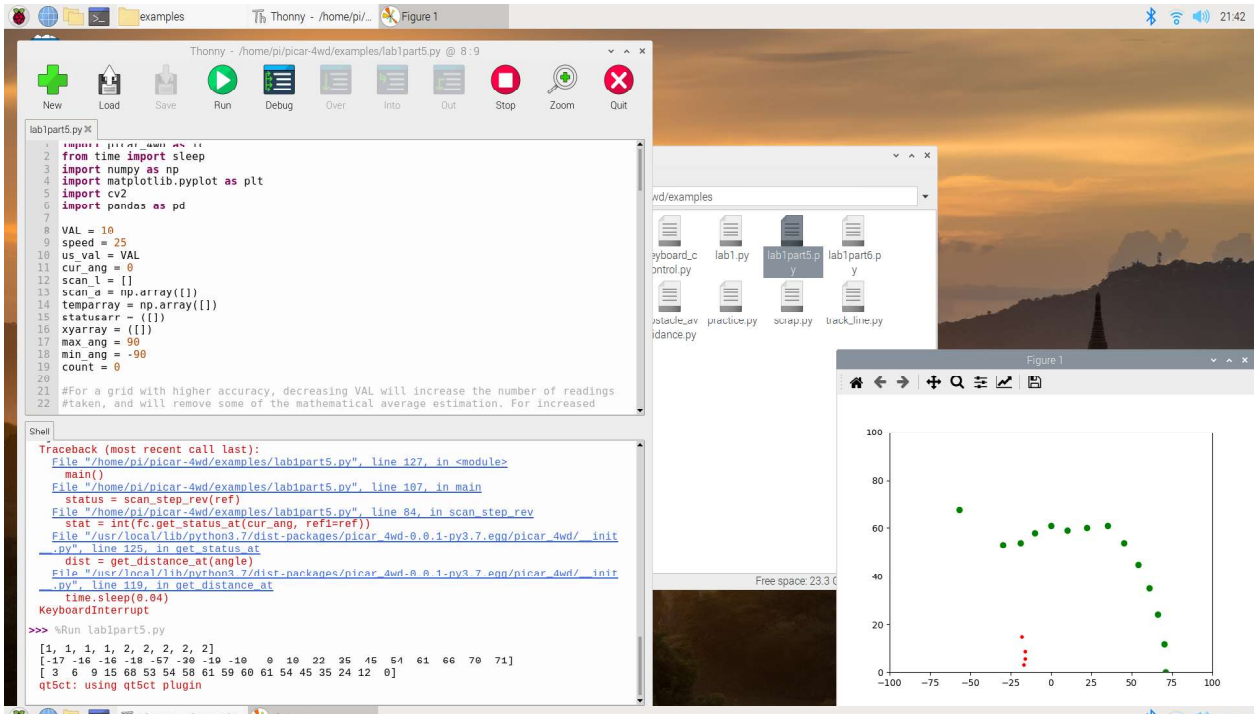
2.1. Advanced Mapping

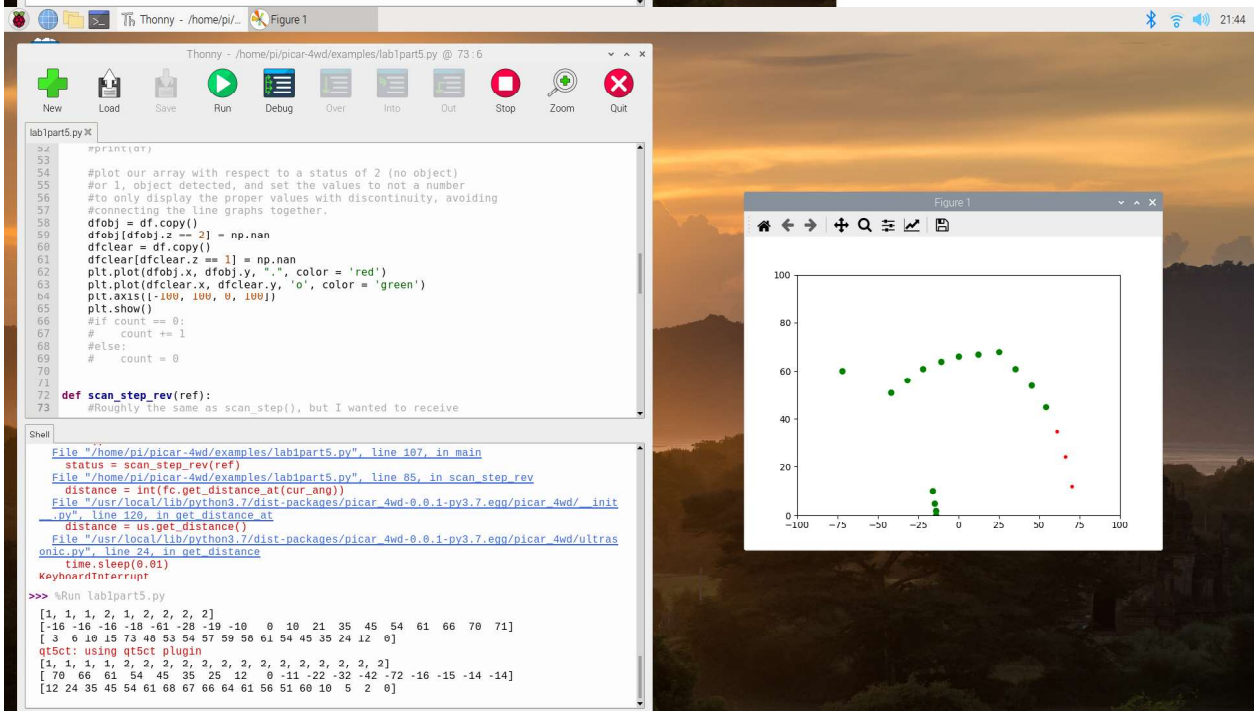
The advanced mapping components became much easier as we (I) learned numpy, and matplotlib congruently through my second course. Learning from the initial success we found in step 3, we were able to rewrite and implement a matplotlib grid which would take distance and angle reading, and convert them to x and y coordinates. From here, the coordinates were prepared through rounding, conversion to integers, and finally sorted into a panda's dataframe. With matplotlib, I was able to display the data onto a graph with some difficulty.

Difficulties arose from three particular areas:

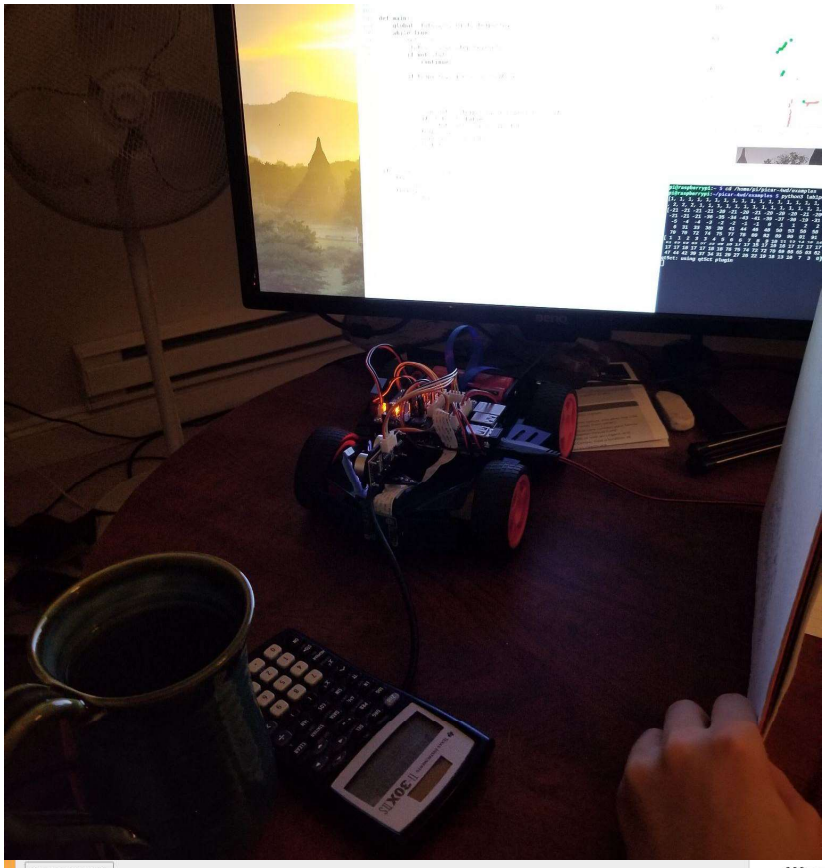
- First, while the x and y coordinates input into the trigonometric functions were correct, I received bizarre output values. The issue turned out to be that numpy considered the values in radians instead of degrees. Further to this point, my x values were displaying opposite to how they should have been interpreted by the car. A simple * -1 solved that issue quickly.
- Second, from not understanding how to properly prepare the data that would be received by the panda's library(https://pandas.pydata.org/docs/user_guide/index.html#user-guide). I ran into a myriad of problems regarding this point, ultimately learning as I went. One such issue was panda's refusal to import a numpy array, which I believe was due to its shape. This was resolved by removing x and y's reshape call, and sending the data to pandas earlier in the code.
- Finally, I struggled greatly with displaying 2 sets of data on one matplotlib graph as the data, when displayed as lines, would connect regardless of if an object or clear space was detected. By learning about, and implementing np.nan to ignore particular values, and display two graphs, this problem was fixed. Finally, my graph would display correctly, half of the time. As a final step, I realized that the status value for my object detection (which was tied to each pair of x and y coordinates) would be incorrect when panning back right to left. By implementing a 0 to 1 count, and flipping z whenever the count was one, I was able to finally resolve the last issue in displaying a graph.

*The first photo shows panning left to right with an object to the cars left. The second photo shows panning right to left, with the same object on the cars left. Through implementing a z flip half the time, this was resolved.


examples
Thonny - /home/pi/...
Figure 1
21:42
Thonny - /home/pi/picar-4wd/examples/lab1part5.py @ 8 : 9
New
Load
Save
Run
Debug
Over
Into
Out
Stop
Zoom
Quit
lab1part5.py
from time import sleep
import numpy as np
import matplotlib.pyplot as plt
import cv2
import pandas as pd
VAL = 10
speed = 25
us_val = VAL
cur_ang = 0
scan_l = []
scan_a = np.array([])
temparray = np.array([])
statusarr = ([])
xyarray = ([])
max_ang = 90
min_ang = -90
count = 0
#For a grid with higher accuracy, decreasing VAL will increase the number of readings
#taken, and will remove some of the mathematical average estimation. For increased
Shell
Traceback (most recent call last):
File "/home/pi/picar-4wd/examples/lab1part5.py", line 127, in <module>
main()
File "/home/pi/picar-4wd/examples/lab1part5.py", line 107, in main
status = scan_step_rev(ref)
File "/home/pi/picar-4wd/examples/lab1part5.py", line 84, in scan_step_rev
stat = int(fc.get_status_at(cur_ang, ref1=ref))
File "/usr/local/lib/python3.7/dist-packages/picar_4wd-0.0.1-py3.7.egg/picar_4wd/__init__.py", line 125, in get_status_at
dist = get_distance_at(angle)
File "/usr/local/lib/python3.7/dist-packages/picar_4wd-0.0.1-py3.7.egg/picar_4wd/__init__.py", line 119, in get_distance_at
time.sleep(0.04)
KeyboardInterrupt
>>> %Run lab1part5.py
[1, 1, 1, 1, 2, 2, 2, 2, 2]
[-17 -16 -16 -18 -57 -30 -10 -10 0 10 22 35 45 54 61 66 70 71]
[3 6 9 15 68 53 54 58 61 59 60 61 54 45 35 24 12 0]
qt5ct: using qt5ct plugin
wd/examples
lab1.py
lab1part5.py
lab1part6.py
practice.py
scrap.py
track_line.py
Free space: 23.3 G
Figure 1
100
80
60
40
20
0
−100
−75
−50
−25
0
25
50
75
100


Thonny - /home/pi/...
Figure 1
21:44
Thonny - /home/pi/picar-4wd/examples/lab1part5.py @ 73 : 6
New
Load
Save
Run
Debug
Over
Into
Out
Stop
Zoom
Quit
lab1part5.py
#print(df)
#plot our array with respect to a status of 2 (no object)
#or 1, object detected, and set the values to not a number
#to only display the proper values with discontinuity, avoiding
#connecting the line graphs together.
dfobj = df.copy()
dfobj[dfobj.z == 2] = np.nan
dfclear = df.copy()
dfclear[dfclear.z == 1] = np.nan
plt.plot(dfobj.x, dfobj.y, ".", color = 'red')
plt.plot(dfclear.x, dfclear.y, 'o', color = 'green')
plt.axis([-100, 100, 0, 100])
plt.show()
#if count == 0:
count += 1
#else:
# count = 0
def scan_step_rev(ref):
#Roughly the same as scan_step(), but I wanted to receive
Shell
File "/home/pi/picar-4wd/examples/lab1part5.py", line 107, in main
status = scan_step_rev(ref)
File "/home/pi/picar-4wd/examples/lab1part5.py", line 85, in scan_step_rev
distance = int(fc.get_distance_at(cur_ang))
File "/usr/local/lib/python3.7/dist-packages/picar_4wd-0.0.1-py3.7.egg/picar_4wd/__init__.py", line 120, in get_distance_at
distance = us.get_distance()
File "/usr/local/lib/python3.7/dist-packages/picar_4wd-0.0.1-py3.7.egg/picar_4wd/ultrasonic.py", line 24, in get_distance
time.sleep(0.01)
KeyboardInterrupt
>>> %Run lab1part5.py
[1, 1, 1, 2, 1, 2, 2, 2, 2]
[-16 -16 -16 -18 -61 -28 -19 -10 0 10 21 35 45 54 61 66 70 71]
[3 6 10 15 73 40 53 54 57 59 56 61 54 45 35 24 12 0]
qt5ct: using qt5ct plugin
[1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
[70 66 61 54 45 35 25 12 0 -11 -22 -32 -42 -72 -16 -15 -14 -14]
[12 24 35 45 54 61 68 67 66 64 61 56 51 60 10 5 2 0]
Figure 1
100
80
60
40
20
0
−100
−75
−50
−25
0
25
50
75
100

To show the accuracy of the mapping, we have also added in photos (in addition to the video) of the map scan at VAL 2 (or 90 values mapped):

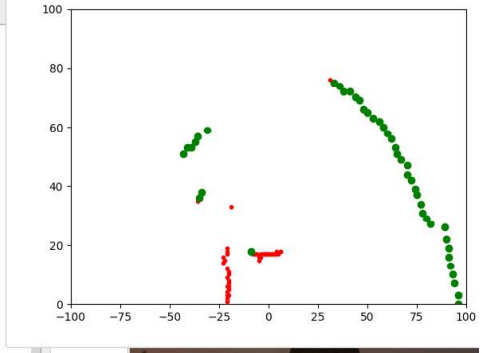


lab1part5.py

```
        return False


def main():
    global statusarr, dist, temparray
    while True:
        ref = 50
        status = scan_step_rev(ref)
        if not status:
            continue

        if temparray.size >= (180/VAL + 2):
            #This is to isolate only the rows of data I want
            #which will be fed into grid(), since the first set
            #only every delivers half the items of data due to its half pan.
            #temparray is only the vehicle to help it escape scan_step_rev
            scan_data = temparray.reshape(int(180/VAL),2)
            if status != False:
                statusarr = np.array(status)
            temparray = ([])
            grid(ref, scan_data)
        print(status)


if __name__ == "__main__":
    try:
        main()
    finally:
        fc.stop()
```

Shell

```
Python 3.7.3 (/usr/bin/python3)
>>>
```



pi@raspberrypi: ~/picar-4wd/examples

File Edit Tabs Help

```
pi@raspberrypi:~ $ cd /home/pi/picar-4wd/examples
pi@raspberrypi:~/picar-4wd/examples $ python3 lab1part5.py
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1,
, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[-21 -21 -21 -21 -20 -21 -20 -21 -20 -20 -20 -21 -20 -20 -21 -23 -22 -23
 -21 -21 -21 -36 -35 -34 -43 -41 -39 -37 -36 -19 -31  -9  -8  -7  -5  -6
  -5  -4  -4  -3  -2  -2  -1  -1   0   1   1   2   2   3   4   4   5   6
   6  31  33  36  38  41  44  46  48  50  53  56  58  60  62  64  65  67
  70  70  72  74  75  77  78  80  82  89  90  91  91  92  93  94  96  96]
[ 1  1  2  3  3  4  5  6  6  7  8  9 10 11 12 14 15 16 17 18 19 35 36 38
 51 53 53 55 57 33 59 18 17 17 15 17 16 16 17 17 17 17 17 17 17 17 17 17
 17 17 18 17 17 18 18 70 75 74 72 72 70 09 06 05 03 02 00 58 50 53 51 49
 47 44 42 39 37 34 31 29 27 26 22 19 16 13 10  7  3  0]
qt5ct: using qt5ct plugin
```

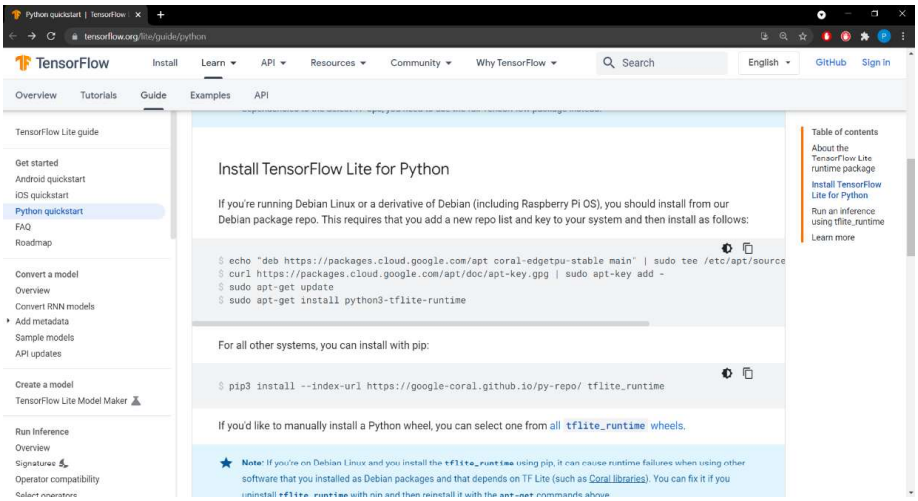
In the first photo, we can see the cars' perspective (and barely make out the map) to show an object immediately to the left, and an object immediately in front, with space between them and the right 180 (roughly 165) degrees wide open. As seen, increased accuracy of the map can be observed by decreasing the distance between scan angles.

2.2.Object Detection

2.2.1. Tensorflow: tflite

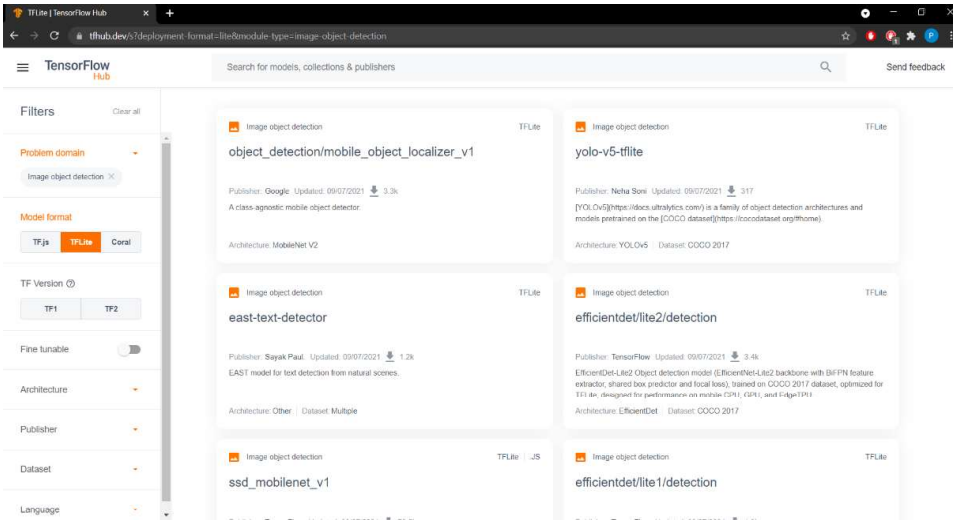
2.2.1.1. Installation with apt-get

https://www.tensorflow.org/lite/guide/python



2.2.1.2. Tflite: Object detection pre-trained models

https://tfhub.dev/s?deployment-format=lite&module-type=image-object-detection



2.2.1.3. Performance metrics on Raspberry Pi (no Hardware accelerator)

TfLite pre-trained models:	Frames per second (Higher the better)	Latency in milliseconds (Lower the better)	Sample accuracy (single frame)
ssd_mobilenet_v3_large.tflite	3.3fps	~300ms	Good
ssd_mobilnet_v3.tflite	8.4fps	~118ms	Poor
detect.tflite	5.4fps	~187ms	Avg

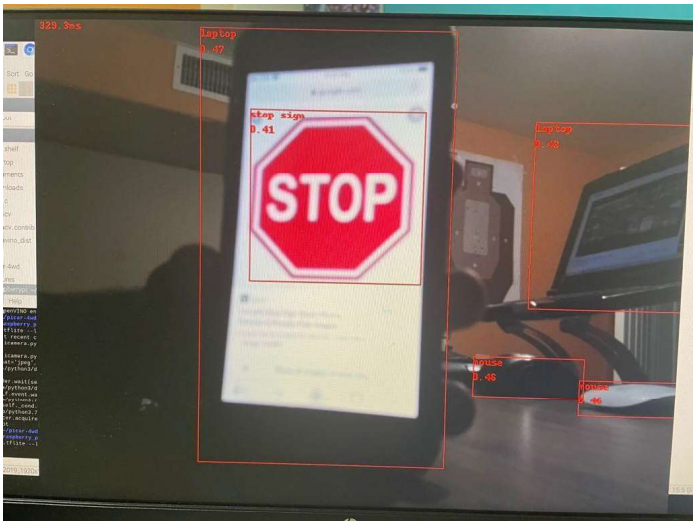


Fig 1: Inference object detection output with "ssd_mobilenet_v3_large.tflite"

Command line args to run object detection with tflite models: Using sample from github

- python3 detect_picamera.py --model ./model/ssd_mobilenet_v3_large.tflite --labels ./model/coco_labels.txt
- python3 detect_picamera.py --model ./model/detect.tflite --labels ./model/coco_labels.txt
- python3 detect_picamera.py --model ./model/ssd_mobilnet_v3.tflite --labels ./model/coco_labels.txt

Note: COCO labels file reference

2.2.2. Inference optimization with Intel Movidius NCS2 (Neural compute stick 2)

2.2.2.1. Introduction: OpenVINO Deep Learning toolkit workflow

Intel OpenVINO™ toolkit:

- Enables CNN-based deep learning inference on the edge
- Supports heterogeneous execution across an Intel® CPU, Intel® Integrated Graphics, Intel® Neural Compute Stick 2 and Intel® Vision Accelerator Design with Intel® Movidius™ VPUs

- Speeds time-to-market via an easy-to-use library of computer vision functions and pre-optimized kernels
- Includes optimized calls for computer vision standards, including OpenCV* and OpenCL™

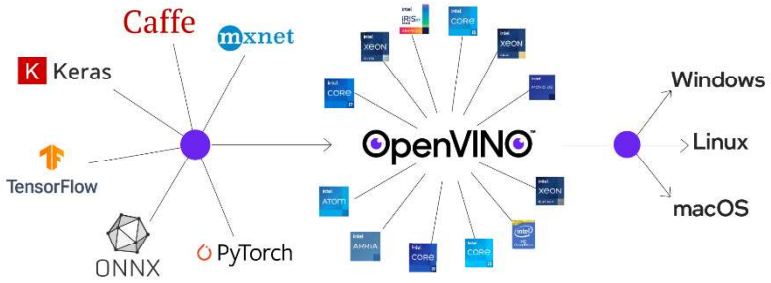


Fig 2: OpenVINO supporting frameworks, OS and Intel CPUs, GPUs, VPUs/FPGAs

OpenVINO Deep Learning toolkit supports optimization and conversion of pre-trained model extensions from popular AI framework (Ex: like Tensorflow, Keras etc.) and converted model is represented in .xml (topology) and .bin (model weights) called as Intermediate Representation (IR). OpenVINO helps accelerate AI inference vision work loads on Intel CPUs, GPUs and VPU, FPGAs.

For this lab, we used *yolo-v2-tiny-tf.xml* OpenVINO IR model (give model details here) from OpenVINO model zoo [github] and run AI vision work load using VPU plugin for Movidius Neural Compute Stick 2 (a USB AI Inference HW accelerator from Intel)

In below Fig 3.you can see work flow of model optimization, fine tuning and quantization support.

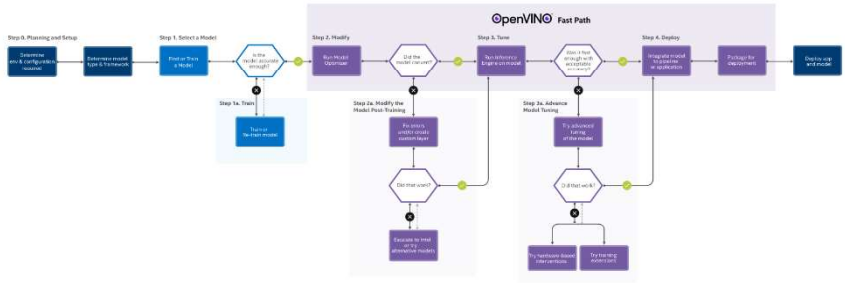


Fig 3: OpenVINO DL Toolkit workflow

2.2.2.2. OpenVINO Installation on Raspberry pi

OpenVINO on Raspberry Pi: Installation Guide
Installer: https://software.seek.intel.com/openvino-toolkit
Build & Run: Object Detection sample

2.2.2.3. Open Model Zoo:

Open model zoo is open source github repository with demo samples and pre-trained Intel (OpenVINO) and Public (Tensorflow/Caffe/Keras etc.) models in various categories like Image classification, Object detection, Segmentation and body pose estimate etc.

For the purpose of traffic sign detection, we considered an Objection Detection demo sample and tested with various pre-trained Intel and Public models trained on COCO dataset.

Github: https://github.com/openvinotoolkit/open_model_zoo

2.2.2.4. Standalone object detection performance with HW accelerator Intel Movidius NCS2

As step one, I started testing performance of standalone object detection models (OpenVINO) on raspberry pi with Neural Compute Stick 2 as HW inference accelerator without any PiCar functionalities. Below are a few OpenVINO models tested as standalone object detection modules and its performance (latency and Frames Per Second). *Yolo-v2-tiny-tf.xml stands out as the best model w.r.t FPS and Latency details*.

OpenVINO demo sample: object_detection_demo.py

OpenVINO IR models:	Frames per second (Higher the better)	Latency in milliseconds (Lower the better) – batch size > 1	Sample accuracy (single frame)
yolo-v2-tiny-tf.xml	13.3fps	~325ms	High
pelee-coco.xml	25fps	~183ms	Poor
yolo-v3-tiny-tf.xml – Model failing	N/A	N/A	N/A

Command line arguments to run Object detection with OpenVINO IR model(s):

- python3 object_detection_demo.py --model /home/pi/Downloads/yolo-v2-tiny/yolo-v2-tiny-tf.xml -d MYRIAD -i /dev/video0 --labels /home/pi/Downloads/MobileNet_SSD/coco_labels.txt --architecture_type=yolo
- python3 object_detection_demo.py --model /home/pi/Downloads/Pelee/pelee-coco.xml -d MYRIAD -i /dev/video0 --labels /home/pi/Downloads/MobileNet_SSD/coco_labels.txt --architecture_type=ssd
- Failing: python3 object_detection_demo.py --model /home/pi/Downloads/yolo-v3-tiny/yolo-v3-tiny-tf.xml -d MYRIAD -i /dev/video0 --labels /home/pi/Downloads/MobileNet_SSD/coco_labels.txt --architecture_type=yolo

2.2.2.5. Optimization:

Even though the standalone performance number looked decent (> 1fps) to performance object detection at 13+fps. After integration Object Detection along with PiCar modules to perform object detection tasks while the car is moving.I saw drastic drop in fps i.e < 1 fps even with inference running on NCS2. After investigating the reason for drop in fps. Below are couple of reasons

- Frame rendering with X11 forwarding to remote session (Putty) had a serious impact on fps /performance.
- Rendering object detected inference frames through OpenCV cv2.imshow slowed down fps drastically

After addressing above slow down issues with code changes - *Achieved real time performance i.e 25fps for object Detection inference with Yolo-tiny v2 using Intel OpenVINO + Moviduis HW Accelerator (NCS2). Note: Details in Section 2.4*

2.2.3. Traffic sign detection

2.2.3.1. Stop sign Detection

Stop sign detection with Yolo V2 Tiny OpenVINO model running at real time on NCS2

FPS: **~24.2fps** **Latency**: 156ms (with batch size > 1) **Confidence**: 91.5%

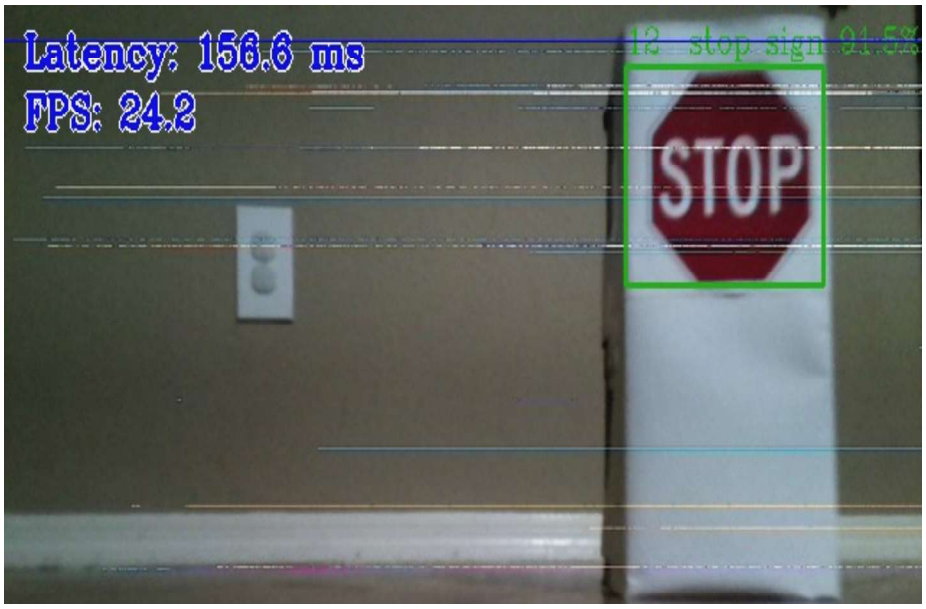
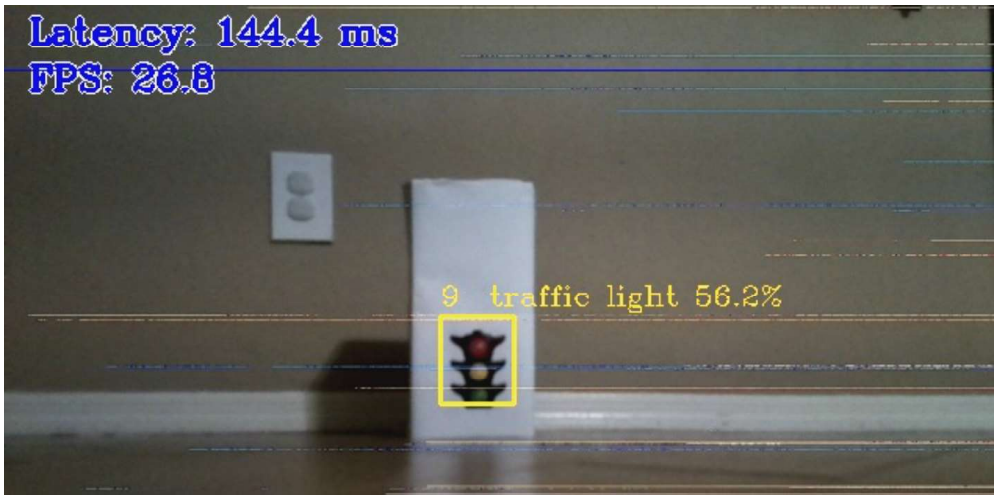


Fig: Stop Sign detection on Raspberry Pi with on-device PiCamera stream recording

2.2.3.2. Traffic light detection:

Traffic light detection with Yolo V2 Tiny OpenVINO model running at real time on NCS2

FPS: **~26.8fps** **Latency**: 144.4s (with batch size > 1) **Confidence**: 56.2%



FYR: Loading of MYRIAD plugins before starting inference

```
pi@raspberrypi:~/Downloads/open_model_zoo/demos/object_detection_demo/python/IOT_Assign1 $ python3 advanced_routing.py

/home/pi/openvino_dist/python/python3.7/ngraph/utils/types.py:37: DeprecationWarning: `np.bool` is a deprecated alias
for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is
safe. If you specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecati
ons
  (NgraphType.boolean, np.bool),
If you want to quit.Please Ctl + C
Loading...Inference Engine...!!!!
[ INFO ] Initializing Inference Engine...
[ INFO ] Loading network...
[ INFO ] Reading network from IR...
[ INFO ] Loading network to MYRIAD plugin...
[ INFO ] Starting inference...
To close the application, press 'CTRL+C' here or switch to the output window and press ESC key
```

2.3. Routing (Self-Driving Navigation)

2.3.1.

Part 7 required several different types of solutions to be implemented to create a viable routing system. These included: active scanning, the ability to measure distance travelled, how to consistently turn at 90 degrees, how the car would orient itself in space, and how the car would react when interacting with obstacles. By creating three different routing programs, and taking successful parts from each, we were able to implement a routing system that could accept and update x and y coordinates, and maneuver to the given destination.

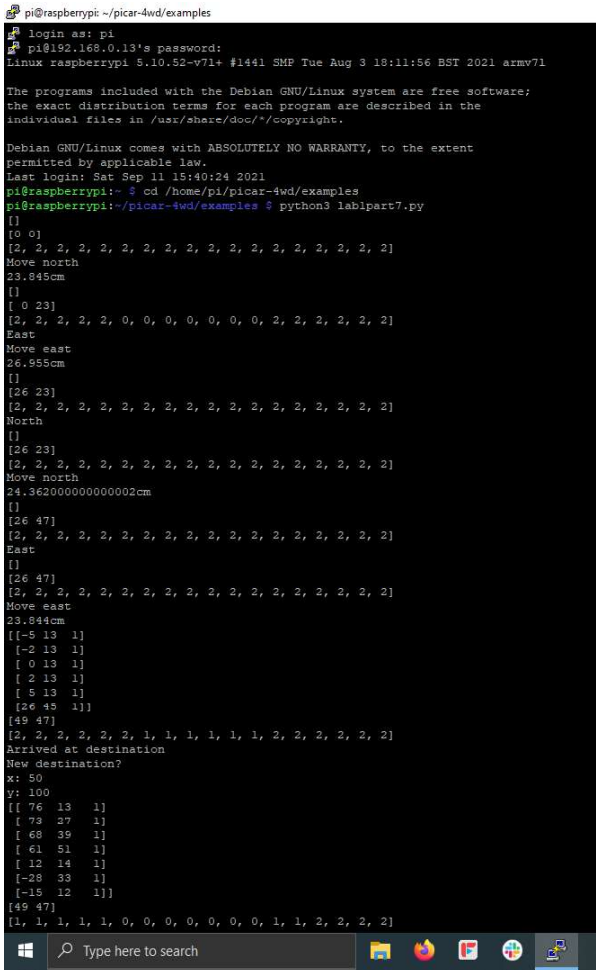
2.3.2. Implementing measurement and x y coordinates

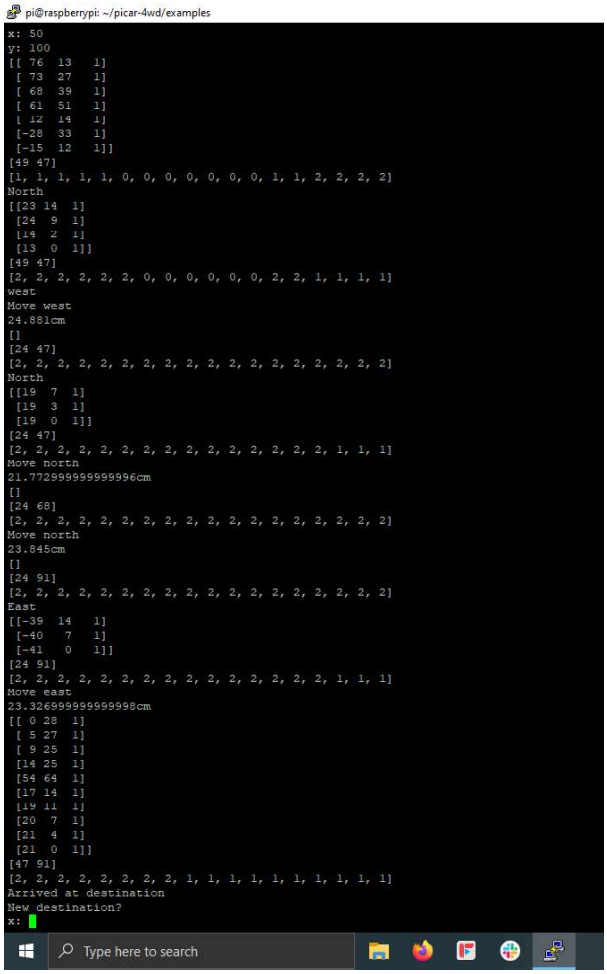
To begin, measuring distance travelled and creating a program that reliably turns at 90 degrees was difficult given the current hardware. Through testing, we were able to have the car travel 25cm forward, and reliably turn left and right at 90 degree angles. Further, now that the car could move, it would need a way to anchor it to a given space, so as to not lose its frame of reference after a single turn. One solution pursued was creating a facing variable that would default to north for each fresh run of the program. As the vehicle would turn and update its direction faced, it was also able to appropriately update its current x and y coordinates with the estimated distance travelled after moving. This allowed us to create a rough framework for how the car would decide to move from here on out. It was determined that y would be the more important coordinate (north and south), and that the car would take in coordinates and compare them to its current location (0,0).

2.3.3. Implementing a routing algorithm

From here, the car would attempt to move forward if the direction was clear. If the direction was not clear, it would rely upon its secondary coordinate, x, and attempted to once again move closer to the destination. The only time it would not actively move towards the destination, is when both coordinates were blocked (which it would then move opposite its secondary coordinate, x, and only travel away from its y destination as a last resort. One method employed to ensure proper obstacle avoidance was to scan after every movement set, and after everytime the car refaced a new direction. This ensured that it would be properly aware of its surroundings, and with object detection set to 30cm away (and its maximal incremental move roughly 25cm), the car had no difficulty with proper avoidance. This system is an excellent intermediate point for creating a robot that can properly scan its surroundings. Our vehicle can successfully route to several destinations, all while actively monitoring its surroundings to avoid stationary and newly placed obstacles.

Photos of the active car routing diagnostics:





The car begins by printing its current location, at which point it will scan its surroundings, and for testing purposes, deliver a status result (to ensure testing accuracy). Once it finishes performing the required trigonometric calculations and deciding which direction is clear, it will choose the appropriate destination to travel to. Once it has completed its move, the car will then output its estimated distance travelled.

The object for destination one, was placed 35cm away from the car to the north. As we can see, the car has detected the object, and knows not only to avoid it, but also to travel to the east (right) since our destination is at (50, 50). Once the car has reached its destination, it prompts the user to input new coordinates to travel to: In this cas (50, 100) or just 2 - 25cm moves to the north.

The car begins facing towards the east, and a new obstacle is placed in front of the car and 10 cm to the left of the car. The car first detects the obstacle in front, and chooses to turn left (as it is the shortest distance away), now once again being blocked facing north, the car turns left once again, and maneuvers around the obstacle placed at (50, 65). It is able to reach its new destination within a matter of 4 turns and 3 moves, prioritizing the shortest path.

2.4. End to End testing:

Advanced Map generation:

Below, fig 2.4.1 illustrates the physical environment of the route with obstacles. We generated a map of this scenario as shown in fig 2.4.2.

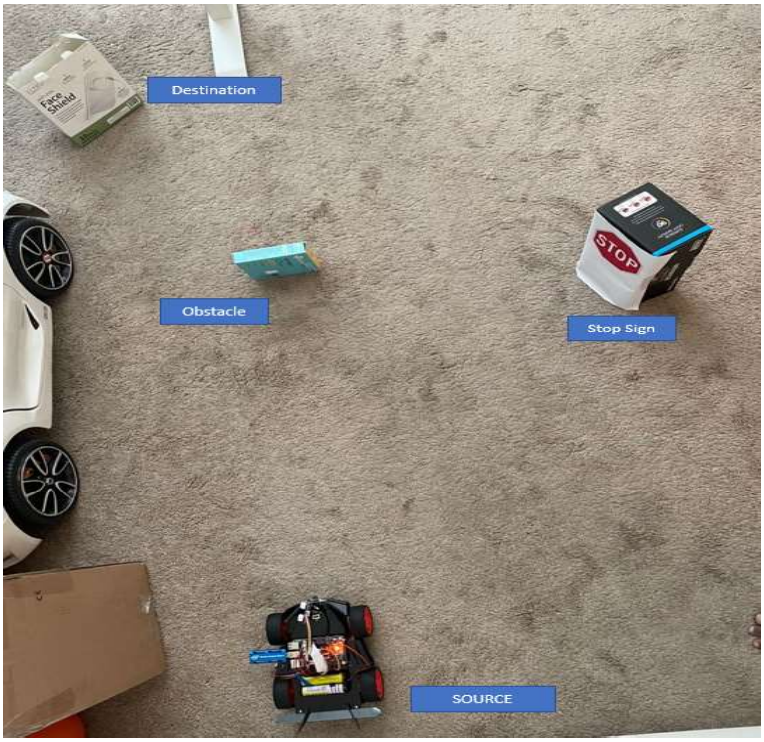


fig 2.4.1: Route layout of physical environment with obstacles

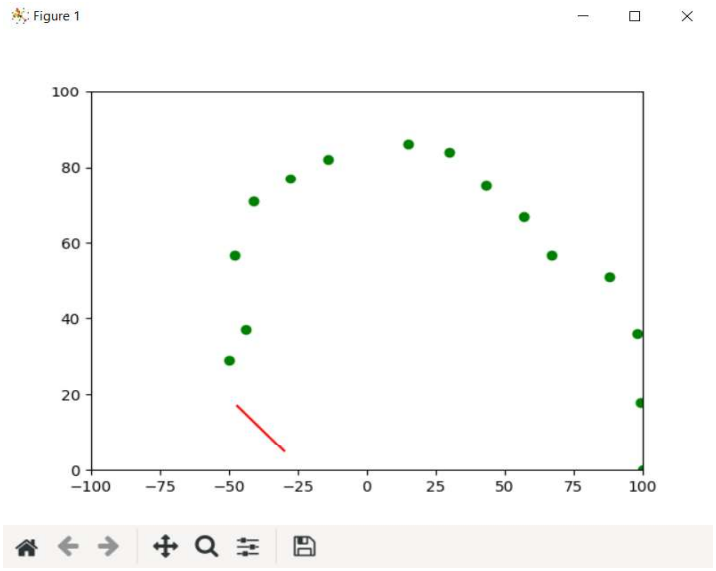


Fig: 2.4.2 - Grid map of above physical environment shown in fig 2.4.1

As you can see, the car is detecting objects that are immediately available within the 30cm range, and regarding objects outside of 30cm as clear. We can tell from the variation of green dots however, that there are obstacles within the map perimeter, but they can be disregarded currently as they are not close enough currently to worry about. The green dots at -25, 70, represent the obstacle, while the dots at 60, 55 represent the stop sign. If we chose to set the scan VAL to a lower step than ever 10 degrees, our map would show even greater definition (at the cost of a much increased scan time).

Object Detection:

As explained in section 2.2, I have used Intel OpenVINO Deep Learning toolkit to accelerate Object Detection using Movidus Neural Compute Stick 2 (ref: fig 2.4.3 for NCS2). This optimization helped meet target fps of ~24 frames per second with object detection using Tiny yolo v2 pre-trained model.

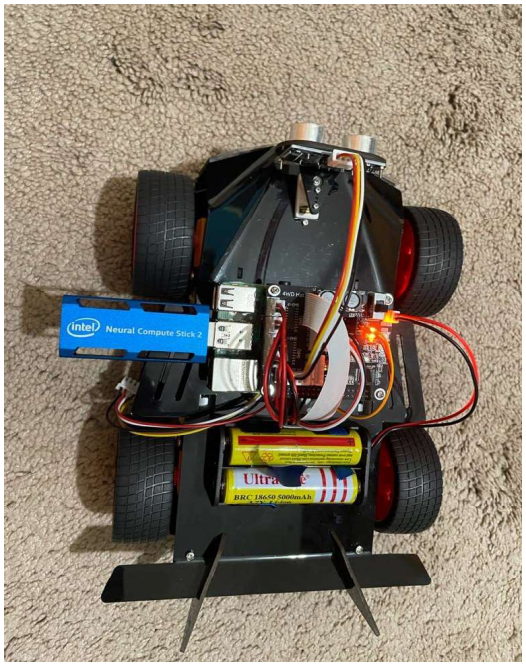


Fig 2.4.3: Raspberry pi with OpenVINO and Neural Compute Stick 2 as HW accelerator

Below snapshot (fig 2.4.4) demonstrates object detection, in this case traffic sign detection running on Movidius Neural Compute Stick 2 with OpenVINO + Yolo V2 tiny IR model.

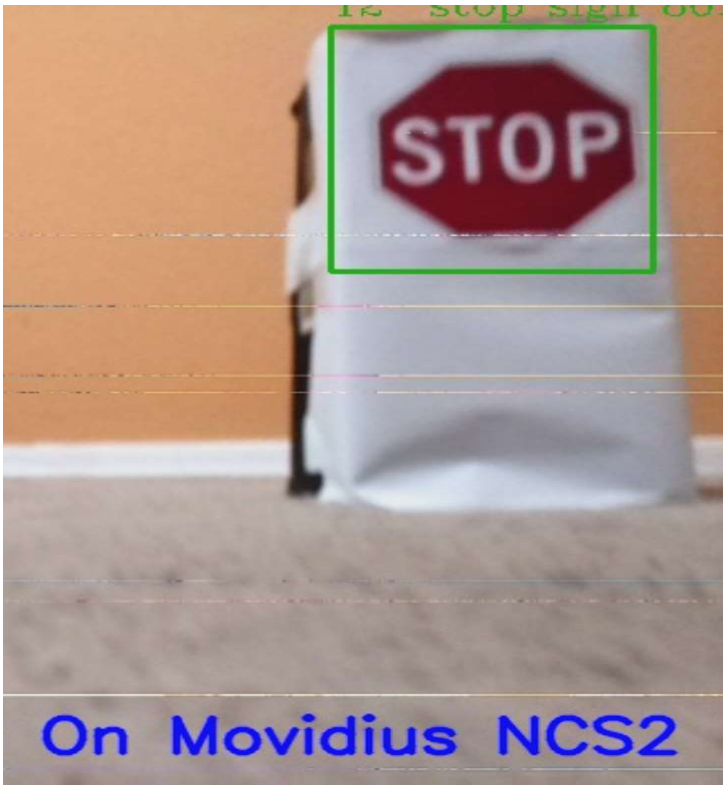


Fig: 2.4.4 - Object detection on NCS2: stop sign detection (bounding box and label) while car navigating along the route to destination [demo in final submission video]

Routing and Testing:

Please refer to the final submission video where we demonstrate the car navigating along the route (path planning) on the grid avoiding obstacles and being able to run object detection of traffic signs at real time ~24fps with Intel OpenVINO + NCS2 HW accelerator.