CS 437 Internet of Things
Lab 1

Group Members:

Peng-Yuan Huang, Guang Hou, Li-Kai Lin, Yeyu Ma

Group Member Contributions:

Peng-Yuan Huang (netID: pyh2)	Step 1: Chassis Assembly; Step 2: Explore the Pycar Code; Step 3: Environment Scanning (Mapping); Step 4: Driving; Lab 1 Part 1 basic mapping and driving code writer; Lab 1 Part 1 demo video provider; Step 6 Object Detection report writer and presenter; Step 8 Testing and Testing demo video provider.
Guang Hou (netID: ghou3)	Step 1: Chassis Assembly; Step 2: Explore the Pycar Code; Step 5: More Advanced Mapping-code, report and video; Step 7: Self-Driving Navigation (Routing)-code report and video.
Li-Kai Lin (netID: likaikl2)	Step 1: Chassis Assembly; Step 2: Explore the Pycar Code; Step 3: Environment Scanning (Mapping); Step 4: Driving; Lab 1 Part 1 demo video provider; Step 6 Object Detection; Step 8 Testing report writer, presenter, and Testing demo video provider.
Yeyu Ma (netID: yeyuma2)	Step 1: Chassis Assembly; Step 2: Explore the Pycar Code; Step 1 - 4 report writer and presenter; Step 5: More Advanced Mapping-code, report; Step 7: Self-Driving Navigation (Routing)-code report.

Due Date:

12:59 AM EST, February 14, 2022

Demo video link:

https://youtu.be/X9p3sZvofbQ
(Backup link:
https://drive.google.com/file/d/1_osqJjeDvDa6l7jlgGOkeefylwdagQvV/view?usp=sharing)

Lab 1 Part 1 Step 1 - 4

Typology of devices

For this lab, we need the components listed below:

- 1x Raspberry Pi Model 4B
- 1x 4WD-Hat module board
- 1x battery holder
- 1x AC adaptor 5.1V 3A
- 2x 18650 Batteries
- 4x DC motors
- 1x Servo
- 1x Ultrasonic Sensor
- 1x Grayscale Sensor
- 1x Photo-interrupter
- 1x Pi Camera v2 module

The Raspberry Pi model 4B is powered by an AC adaptor while the device is stationary. It's useful for the scenario when the group needs to have a quick code fix on the Pi without waiting for batteries to be charged (refer to Wiring Diagram 7).

The 4WD-Hat module is connected to the Pi board via a pair of pin headers based on the assembling instruction from sunfounder (refer to Wiring Diagram 1).

Two 18650 Batteries are used to power both the Raspberry Pi model and the 4WD-Hat module board when the components are assembled and running experiments. A battery holder for the batteries is directly connected to the 4WD-Hat module via a pair of JST connectors (refer to Wiring Diagram 2).

Four DC motors are connected to the 4WD-Hat module board via JST connectors (refer to Wiring Diagram 3).

The servo, the grayscale sensor and the ultrasonic sensors are connected to the 4WD-Hat module board via JST connectors as well. The input power of the servo is 5V, while the grayscale sensor and the ultrasonic sensors' VCC are both 3.3V (refer to Wiring Diagram 4).

The photo-interrupter is connected to the 4WD-Hat module board via JST connectors as well. Its VCC is 3V (refer to Wiring Diagram 5).

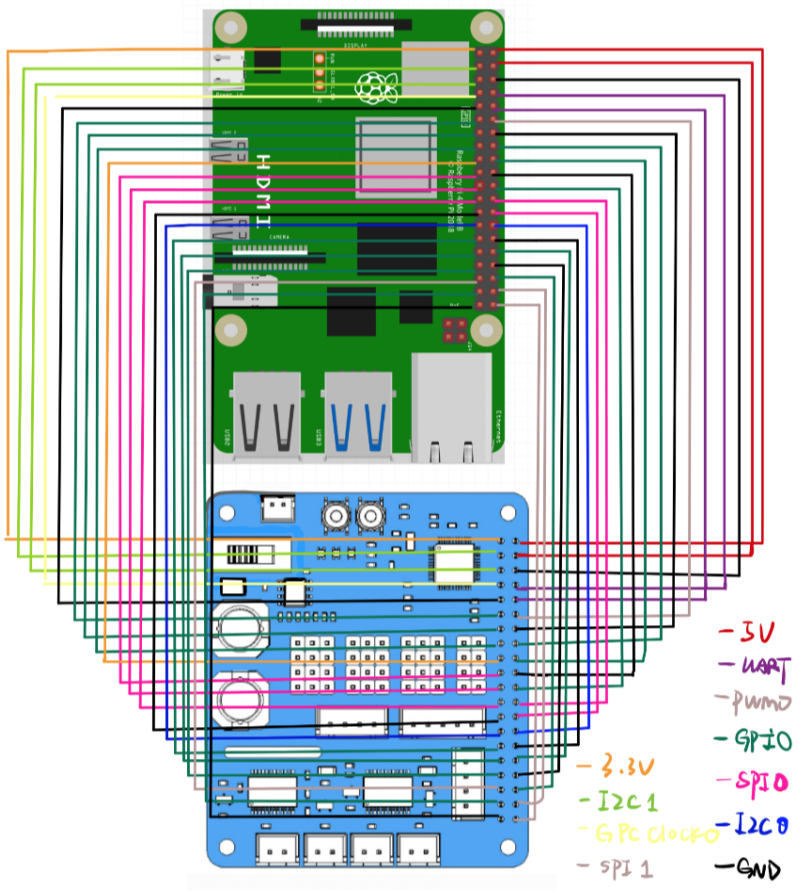
The Pi Camera v2 module is connected to Raspberry Pi board via a 15–pin flex cable (refer to Wiring Diagram 6).

Table 1: Pin mappings for Actuators and Sensors to 4WD-Hat module to Raspberry Pi

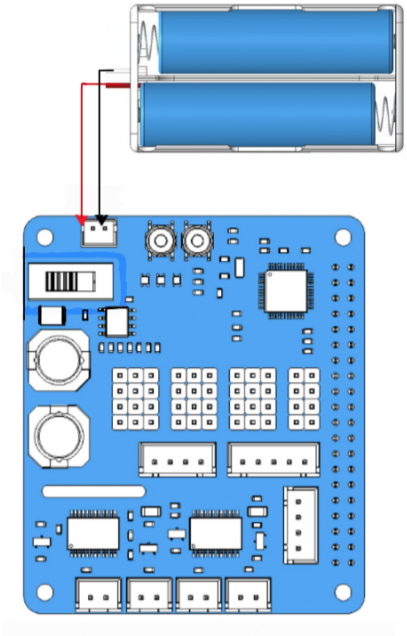
Component	**Wiring/Pin Labeled on component**	**Pin Labeled on 4WD-Hat module board**	**Pin on Raspberry Pi board**
Motor 1	Red Wire	Motor 1- D4	I2C(GPIO2, GPIO3) GPIO23
Motor 2	Red Wire	Motor 2 - D5	I2C(GPIO2, GPIO3) GPIO24
Motor 3	Red Wire	Motor 3 - D11	I2C(GPIO2, GPIO3) GPIO13
Motor 4	Red Wire	Motor 4 - D15	I2C(GPIO2, GPIO3) GPIO20
Servo	Orange Wire	PWM0	I2C(GPIO2, GPIO3)
GreyScale Sensor	S0 S1 S2	A5 A6 A7	I2C(GPIO2, GPIO3)
Ultrasonic Sensor	Trig Echo	D8 D9	GPIO5 GPIO6
Photo-interrupter	S0 S1	D7 D6	GPIO4 GPIO25

Reference links: :

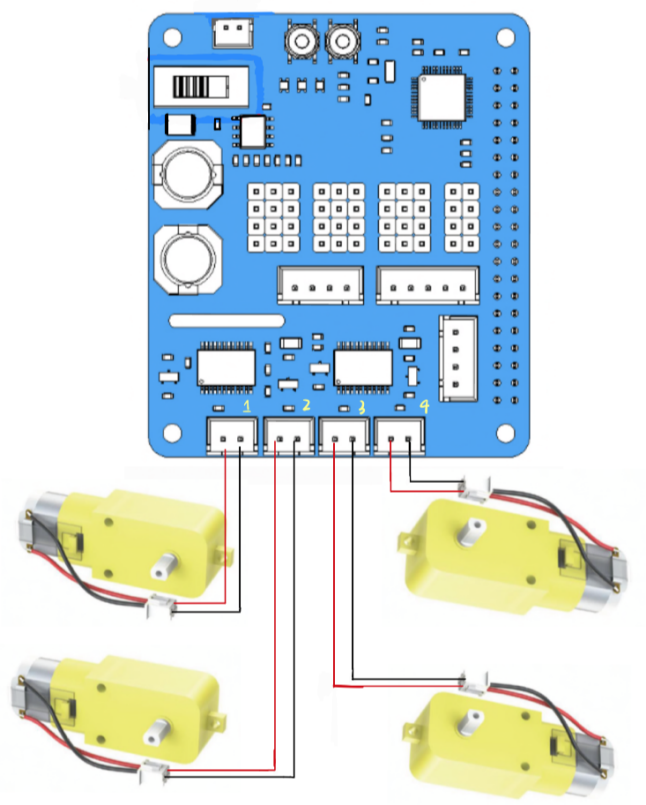
- https://www.theengineeringprojects.com/wp-content/uploads/2021/03/raspberry-pi-4.png
- https://github.com/sunfounder/picar-4wd/blob/698a7677c645adc222637f8c94d6e24b8d83a9e3/picar_4wd/__init__.py
- https://github.com/sunfounder/picar-4wd/blob/698a7677c645adc222637f8c94d6e24b8d83a9e3/picar_4wd/pin.py



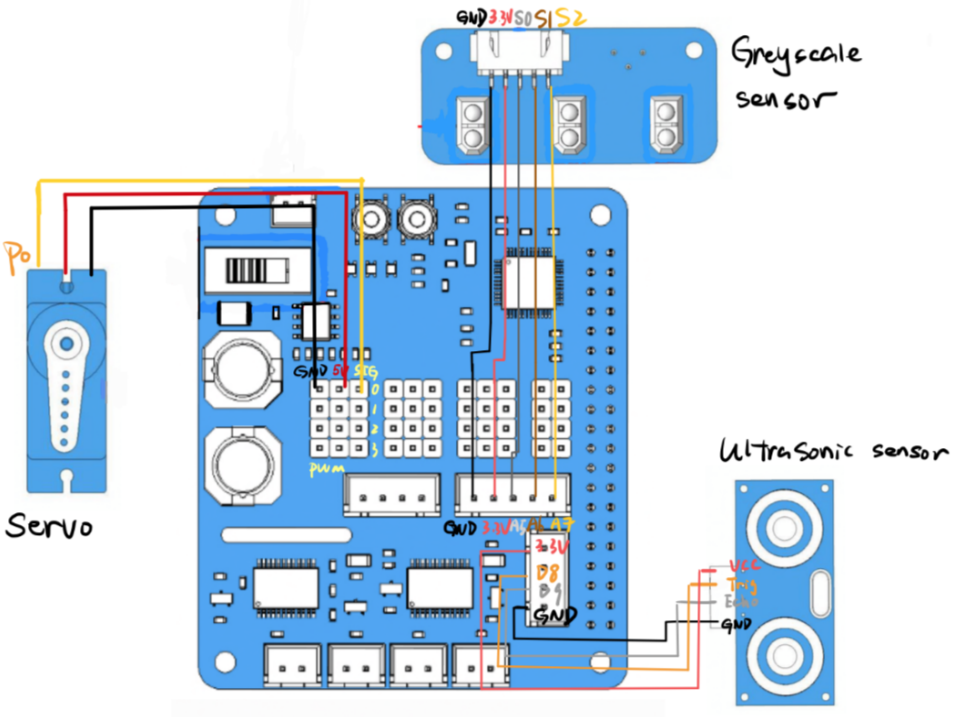
Wiring Diagram 1: 4WD-Hat to Raspberry Pi connected by 40- pin-cables on board



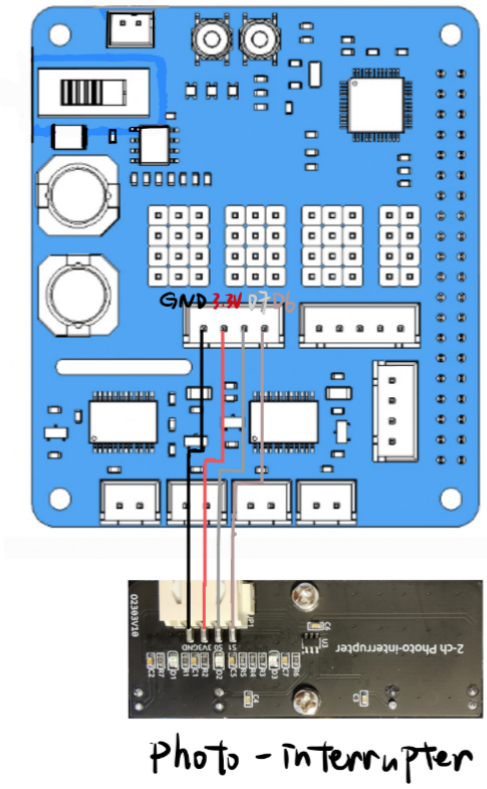
Wiring Diagram 2: Battery Holder to 4WD-Hat



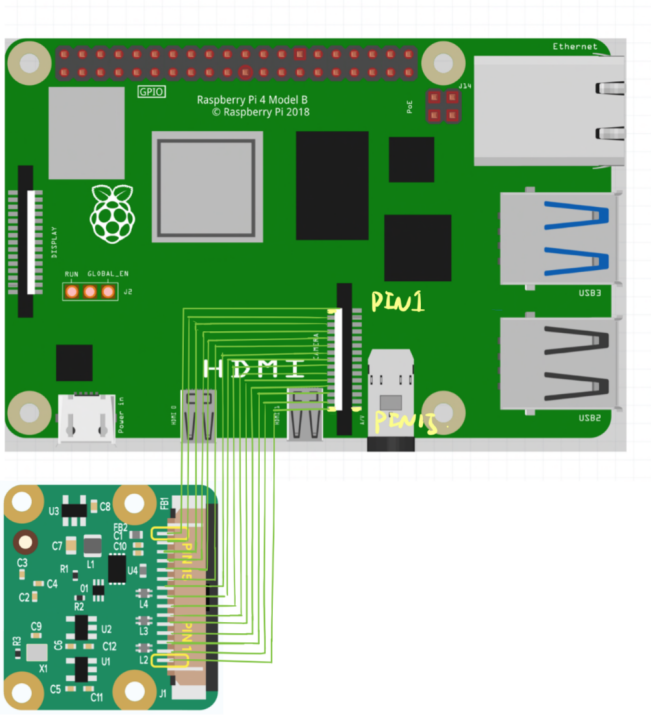
Wiring Diagram 3: Motors to 4WD-Hat



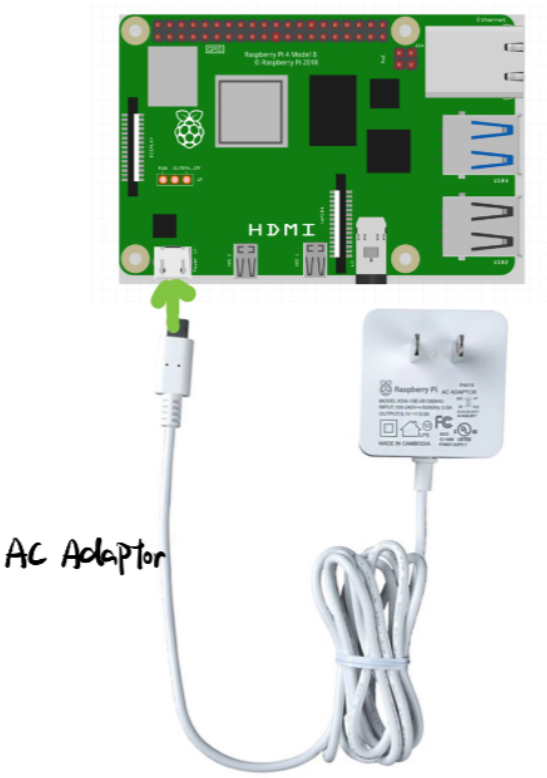
Wiring Diagram 4: Greyscale Detector, Ultrasonic Sensor, and Servo to 4WD-Hat



Wiring Diagram 5: Photo interrupter to 4WD-Hat



Wiring Diagram 6: Pi Camera v2 to Raspberry Pi



Wiring Diagram 7: External Power to Raspberry Pi

Design considerations for Part 1

In part 1, the team uses ultrasonic and servo together as a combination to measure the distance from any obstacles to the pi car itself. The ultrasonic sensor takes samples for 1 times in total (once every 18°) when the servo makes rotations.Then the pi makes judgement on the distance states of the 4th and the 7th sample (i.e the elements in the **scan_focus** list in the code). The moving speed of the car is set to 40 currently in the code. If the ultrasonic sensor indicates no obstacle, then the **scan_focus** list contains all values as 2. The car continues to move forward, and it stops for 0.3 seconds before going to the next checking cycle. This short-interval stop is added to prevent collisions between the pi car and any objects whose movement may not be caught by the ultrasonic sensor due to the sensor accuracy issues.

```
pi@raspberrypi:~/picar-4wd/examples $ python3 part1.py
[2, 2, 2, 2]
[2, 2, 2, 2]
```

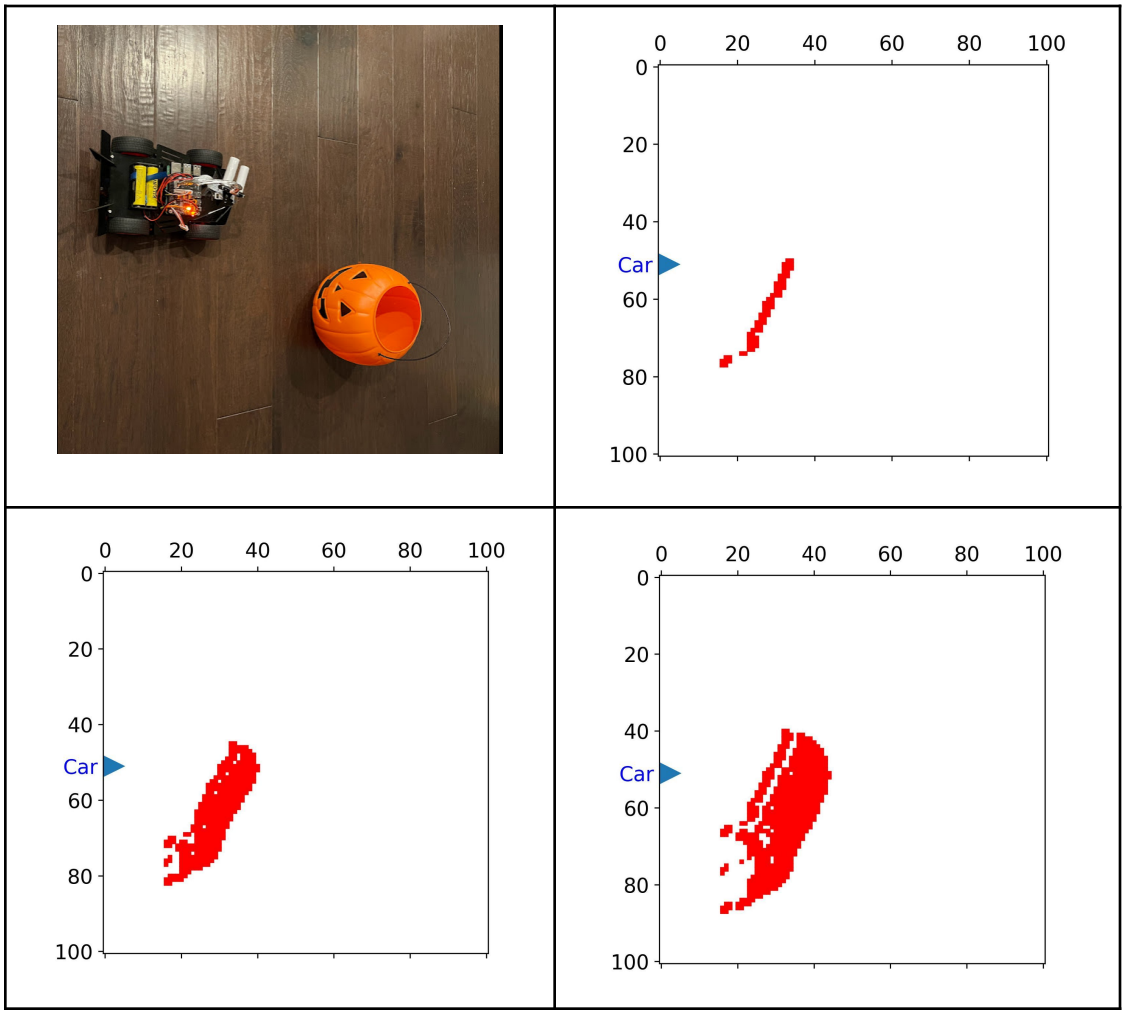
Diagram 8: The outputs as an example of scan_focus list with all values of 2

If the sensor finds any obstacles, i.e any of the returned values in the list is not 2, the car moves back forward first and it pauses for 0.3 seconds to make a complete stop. Then it will turn left and pause for another 0.2 seconds before going to the next checking cycle. The short pause is added for the same reason mentioned above to prevent collision from unexpected moving objects.

Demo video for Part 1
https://www.youtube.com/watch?v=lS_E9ROlVK4
This demo video is for part 1, which shows exactly what is explained in **Design considerations for Part 1.**

Lab 1 Part 2 Step 5 - 8

Step 5: More Advanced Mapping



Top left: car and obstacle positions. Top right: advanced mapping with 0 cm clearance. Bottom left: advanced mapping with 5 cm clearance. Bottom right: advanced mapping with 10 cm clearance.

The car uses the ultrasonic sensor to map the surroundings. The sensor measures every 5 degrees and returns the measured (angle, distance) tuple if there is an object at the specific angle.

To account for measurement error, standalone points are removed. A point with non-zero distance measurement is defined as a standalone point if both the distance at previous angle before this point and the following angle after this point.
A numpy array **scan_map** is used to convert the (angle, distance) data to a 2D map. The data is coveted from polar coordinates to cartesian coordinates. Suppose the car location is (xOrigin, yOrigin), a measurement of (angle, distance) will be converted to (xPos, yPos):

```
        xPos = distance * math.sin(angle) + xOrigin
        yPos = distance * math.cos(angle) + yOrigin
```

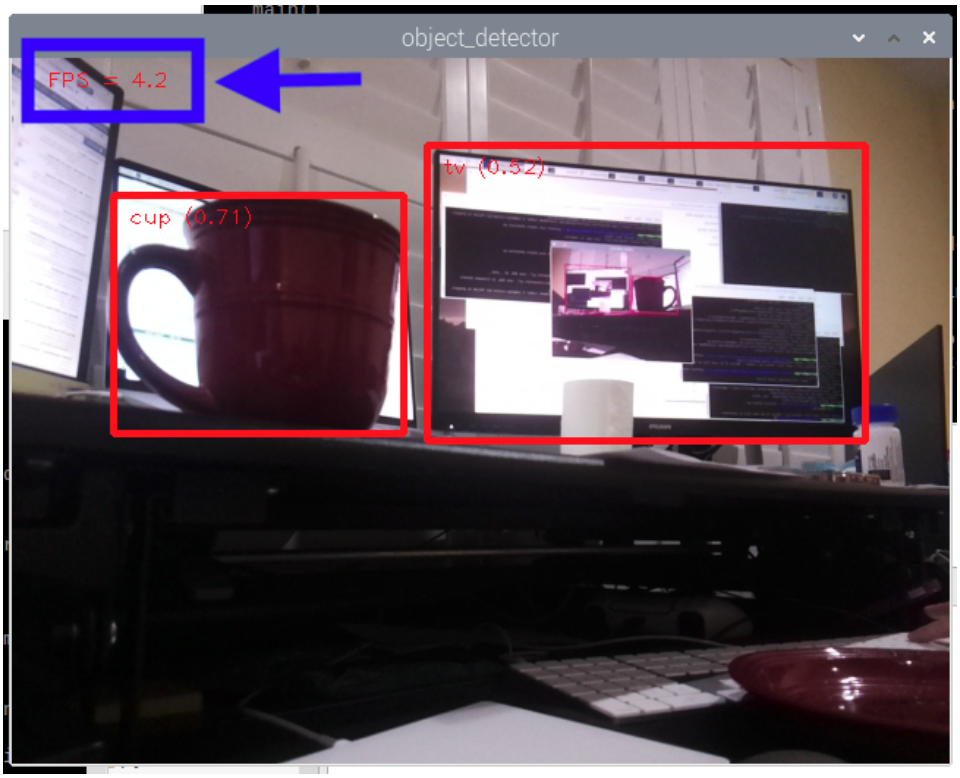
If distance data is valid, the array scan_map[xPos][yPos] is labeled as 1. All other points are labeled as 0 as default.

For improved detection of objects, extra points are filled in between valid points. A line is calculated between the two points. For the intermediate points between the valid points, the nearest ones are labeled as 1.

To help for object avoidance in the routing. A **clearance** is added to the map. For example if **clearance = 10 cm**, the edge of detected objects are extended 5 cm outwards. Using this technique, the added clearance helps to consider the size of the car body. The optimal path can be used directly to control the car without further considering the collision between the car and the obstacles.

Step 6: Object Detection

- Increase FPS



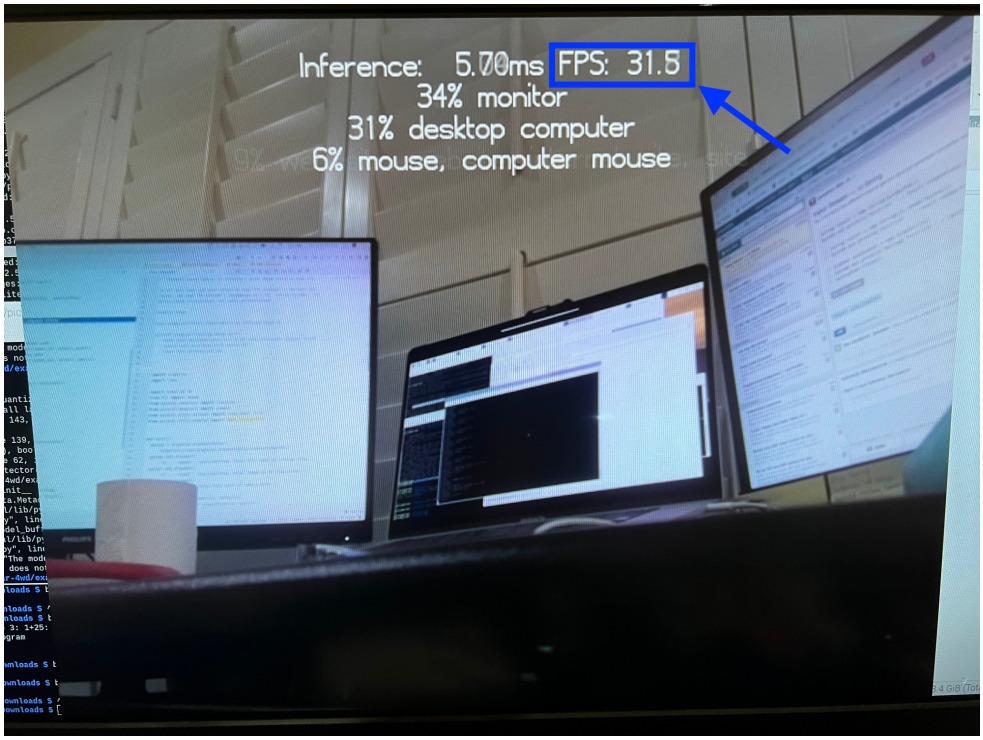
Original FPS using PiCar and OpenCV to detect objects: 4.2 frames per second

As we can see from the above screenshot, when using the PiCar camera and OpenCV with a pre-trained CNN model provided by Tensorflow loaded to perform object detection, originally, although it could detect and recognize the objects, the FPS was as low as 4.2. Therefore, the first challenge we encountered was to improve FPS while PiCar was performing image processing.

Our first option was getting a USB accelerator, like the Coral USB Accelerator, which is probably the only working USB accelerator on the market and this is also suggested by Tensorflow (Link). According to the website description, it "adds an Edge TPU coprocessor to your system, enabling high-speed machine learning inferencing on a wide range of systems"[1]. Once we attach it to the PiCar, the accelerator should help "significantly speed up the inference time"[2]. Later we will discuss more options to improve the FPS while PiCar is in motion.

---

[1] Reference: https://coral.ai/products/accelerator
[2] Reference: https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/raspberry_pi#speed-up-model-inference-optional



Improved FPS by attaching a USB accelerator to PiCar: 31.5 frames per second

As a result, the USB accelerator did help increase the FPS by almost 8 times (4.2 -> 31.5), and it almost reached real-time video processing. So for the next step, we would start training our own model to recognize the stop sign, the red traffic light, the green traffic light, the railroad sign, the speed limit 25 & 40, pedestrians, and an obstacle.

- Train traffic signs detection model - First attempt (Tensorflow + Google Colab)

Example training image for model training

To train our own model using Tensorflow, our first try was following this tutorial[3]. For preparation, we took 77 images similar to the above image and put the objects and traffic signs randomly in each image. We then labeled each image with the bounding box for each object on the image. After that, we selected the MobileNet v2 SSD COCO **Quantized** model to work on because Edge TPU can only run quantized models and set the training steps to 2000.

```
INFO:tensorflow:Performing evaluation on 7 images.
I0127 09:23:02.978426 140392708462336 coco_evaluation.py:293] Performing evaluation on 7 images.
creating index...
index created!
INFO:tensorflow:Loading and preparing annotation results...
I0127 09:23:02.979414 140392708462336 coco_tools.py:116] Loading and preparing annotation results...
INFO:tensorflow:DONE (t=0.00s)
I0127 09:23:02.980530 140392708462336 coco_tools.py:138] DONE (t=0.00s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.09s).
Accumulating evaluation results...
DONE (t=0.04s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
INFO:tensorflow:Finished evaluation at 2022-01-27-09:23:03
I0127 09:23:03.802853 140395047593856 evaluation.py:275] Finished evaluation at 2022-01-27-09:23:03
```

[3] Reference: https://colab.research.google.com/github/dctian/DeepPiCar/blob/master/models/object_detection/code/tensorflow_traffic_sign_detection.ipynb

```
I0127 14:16:19.212467 140395047593856 estimator.py:2109] Saving 'checkpoint_path' summary for global step 2000: /content/gdrive/My Drive/Colab Notebooks/TransferLearning/Training
INFO:tensorflow:Performing the final export in the end of training.
I0127 14:16:19.213874 140395047593856 exporter.py:410] Performing the final export in the end of training.
INFO:tensorflow:Calling model_fn.
I0127 14:16:19.549181 140395047593856 estimator.py:1148] Calling model_fn.
INFO:tensorflow:depth of additional conv before box predictor: 0
I0127 14:16:22.121269 140395047593856 convolutional_box_predictor.py:156] depth of additional conv before box predictor: 0
INFO:tensorflow:depth of additional conv before box predictor: 0
I0127 14:16:22.159411 140395047593856 convolutional_box_predictor.py:156] depth of additional conv before box predictor: 0
INFO:tensorflow:depth of additional conv before box predictor: 0
I0127 14:16:22.197197 140395047593856 convolutional_box_predictor.py:156] depth of additional conv before box predictor: 0
INFO:tensorflow:depth of additional conv before box predictor: 0
I0127 14:16:22.234025 140395047593856 convolutional_box_predictor.py:156] depth of additional conv before box predictor: 0
INFO:tensorflow:depth of additional conv before box predictor: 0
I0127 14:16:22.271929 140395047593856 convolutional_box_predictor.py:156] depth of additional conv before box predictor: 0
INFO:tensorflow:depth of additional conv before box predictor: 0
I0127 14:16:22.309831 140395047593856 convolutional_box_predictor.py:156] depth of additional conv before box predictor: 0
INFO:tensorflow:Done calling model_fn.
I0127 14:16:23.411352 140395047593856 estimator.py:1150] Done calling model_fn.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/saved_model/signature_def_utils_impl.py:201: build_tensor_info (from tensorflow.python.saved_model.uti
Instructions for updating:
This function will only be available through the v1 compatibility library as tf.compat.v1.saved_model.utils.build_tensor_info or tf.compat.v1.saved_model.build_tensor_info.
W0127 14:16:23.411795 140395047593856 deprecation.py:323] From /tensorflow-1.15.2/python3.7/tensorflow_core/python/saved_model/signature_def_utils_impl.py:201: build_tensor_info
Instructions for updating:
This function will only be available through the v1 compatibility library as tf.compat.v1.saved_model.utils.build_tensor_info or tf.compat.v1.saved_model.build_tensor_info.
INFO:tensorflow:Signatures INCLUDED in export for Classify: None
I0127 14:16:23.412785 140395047593856 export_utils.py:170] Signatures INCLUDED in export for Classify: None
INFO:tensorflow:Signatures INCLUDED in export for Regress: None
I0127 14:16:23.412950 140395047593856 export_utils.py:170] Signatures INCLUDED in export for Regress: None
INFO:tensorflow:Signatures INCLUDED in export for Predict: ['tensorflow/serving/predict', 'serving_default']
I0127 14:16:23.413060 140395047593856 export_utils.py:170] Signatures INCLUDED in export for Predict: ['tensorflow/serving/predict', 'serving_default']
INFO:tensorflow:Signatures INCLUDED in export for Train: None
I0127 14:16:23.413145 140395047593856 export_utils.py:170] Signatures INCLUDED in export for Train: None
INFO:tensorflow:Signatures INCLUDED in export for Eval: None
I0127 14:16:23.413225 140395047593856 export_utils.py:170] Signatures INCLUDED in export for Eval: None
INFO:tensorflow:Restoring parameters from /content/gdrive/My Drive/Colab Notebooks/TransferLearning/Training/model.ckpt-2000
I0127 14:16:23.418558 140395047593856 saver.py:1284] Restoring parameters from /content/gdrive/My Drive/Colab Notebooks/TransferLearning/Training/model.ckpt-2000
INFO:tensorflow:Assets added to graph.
I0127 14:16:24.028427 140395047593856 builder_impl.py:665] Assets added to graph.
INFO:tensorflow:No assets to write.
I0127 14:16:24.028689 140395047593856 builder_impl.py:460] No assets to write.
INFO:tensorflow:SavedModel written to: /content/gdrive/My Drive/Colab Notebooks/TransferLearning/Training/export/Servo/temp-b'1643292979'/saved_model.pb
I0127 14:16:25.145556 140395047593856 builder_impl.py:425] SavedModel written to: /content/gdrive/My Drive/Colab Notebooks/TransferLearning/Training/export/Servo/temp-b'164329297
INFO:tensorflow:Loss for final step: 3.1457055.
I0127 14:16:26.899348 140395047593856 estimator.py:371] Loss for final step: 3.1457055.
```

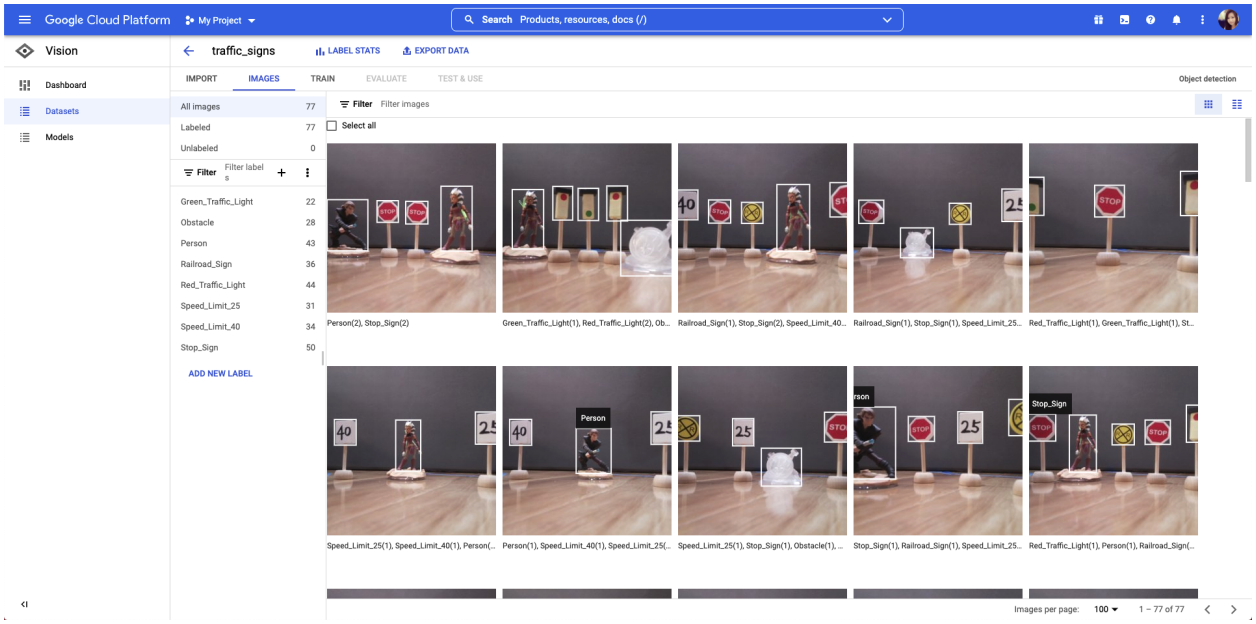
Log information when running the model training session

We ran the model training session three times (1st: 2000 steps/5 hrs; 2nd: 3000 steps/8 hrs; 3rd: 5000 steps/13 hrs) but unfortunately, none of them was a success. At first, we thought it might be because we didn't set up enough training steps but then we noticed the average precision was always 0.000 and after doing some research online, it wasn't a correct performance. Some who had the similar issue[4] suggested it might be because we incorrectly generated the *tfrecord* for the training. However, due to lack of machine learning background knowledge, we were unsure of how to fix the problem. So we went searching for another way to train the models.

- Train traffic signs detection model - Second attempt (Google Cloud Platform + AutoAI)

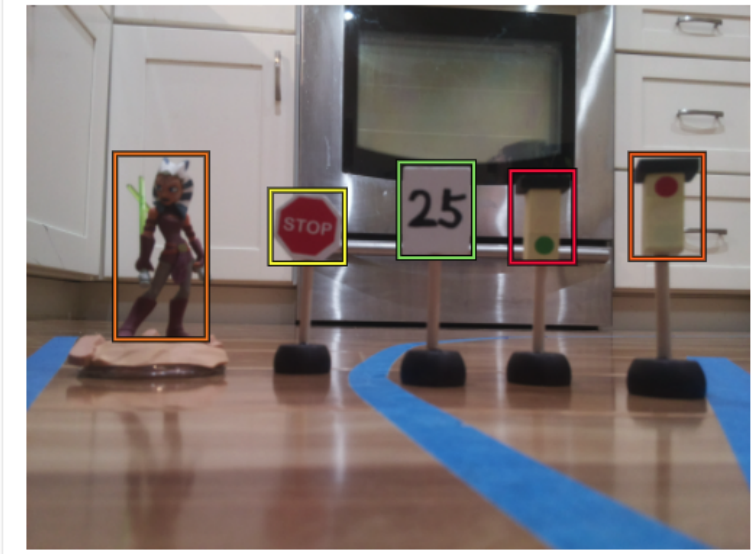
Alternatively, we found Google Cloud Platform also provides model training services and all we needed to do was upload the images and label them. After that, GCP would do the rest of the hard work for us.

[4] Reference: https://github.com/tensorflow/models/issues/7423



Using GCP to train the model

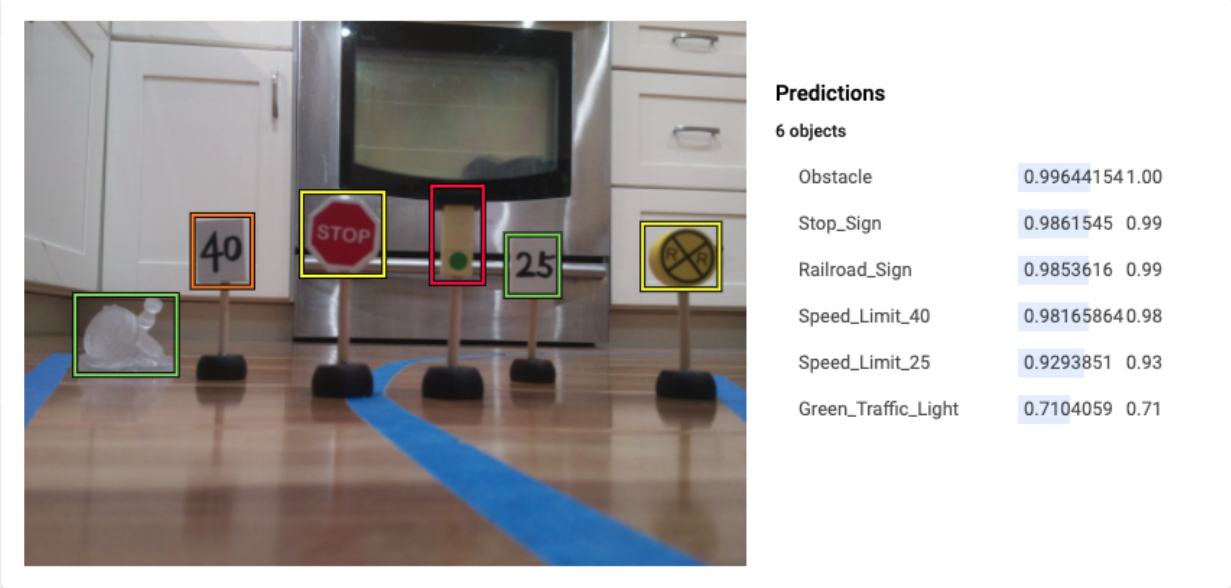
Results



Predictions

5 objects

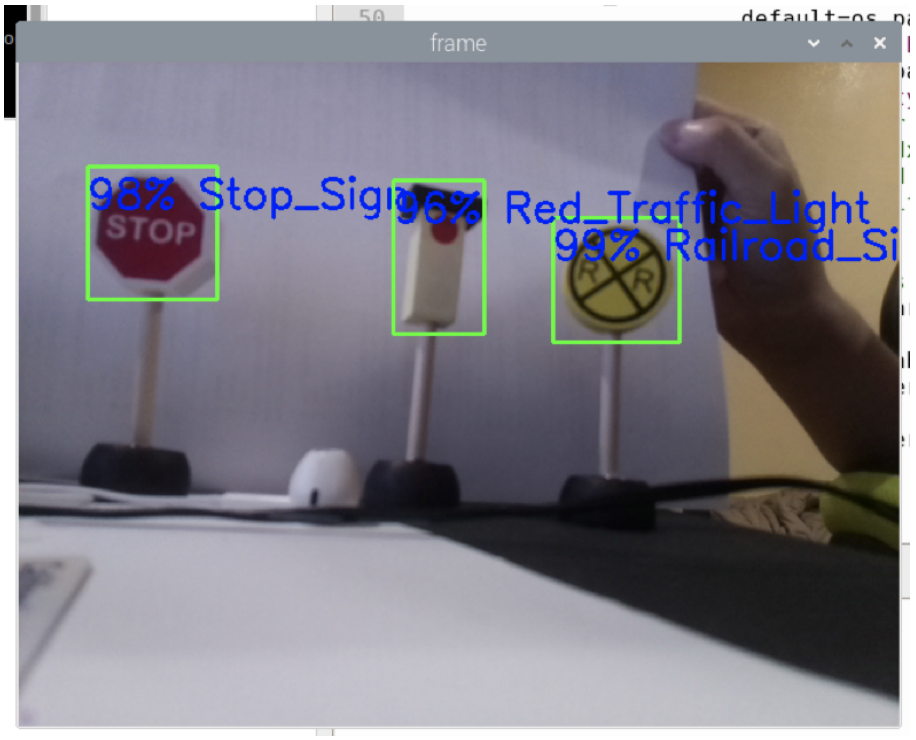
Speed_Limit_25	0.9756311	0.98
Stop_Sign	0.96529126	0.97
Person	0.9165788	0.92
Red_Traffic_Light	0.90971416	0.91
Green_Traffic_Light	0.9076144	0.91



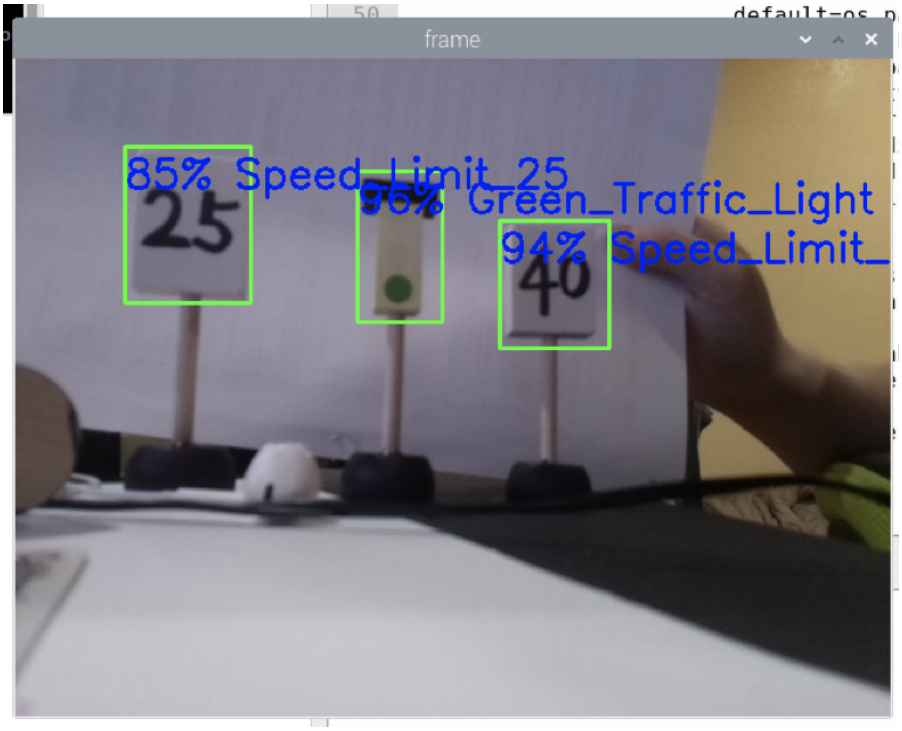
GCP test results after the training was completed

From the above screenshots we can see the training was successful and it had pretty good precision on object predictions/detections. Next, we could finally test the model to see if it would also work on our PiCar.

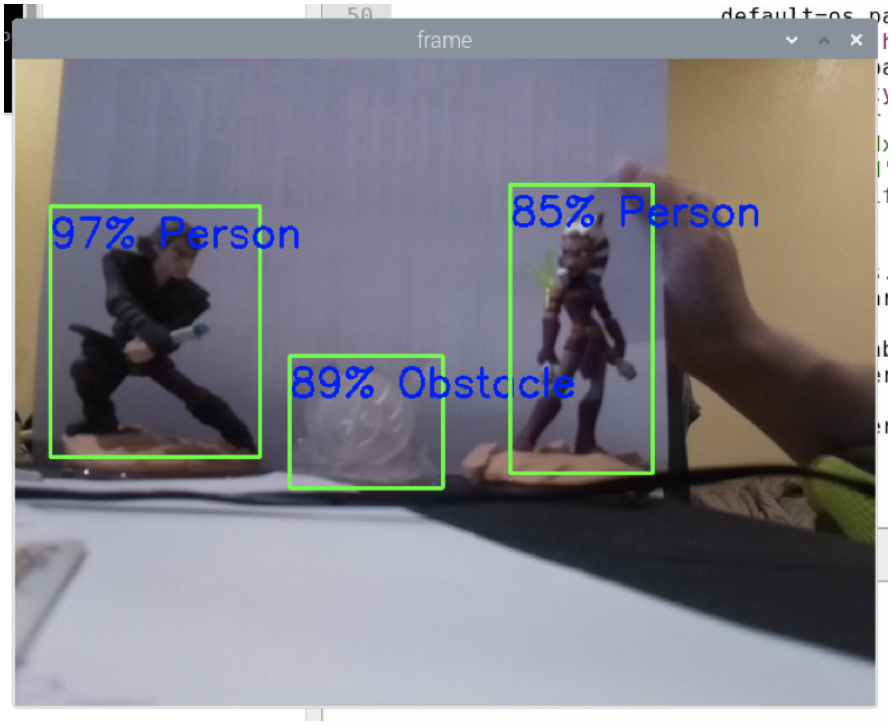
Test on Picar



Traffic sign and object detection on PiCar: Stop Sign, Red Traffic Sign, and Railroad Sign



Traffic sign and object detection on PiCar: Speed Limit 25, Green Traffic Light, Speed Limit 40



Traffic sign and object detection on PiCar: Person (Pedestrian) and Obstacle

As a result, our PiCar successfully detected each traffic sign and object with correct labels. Now we could experiment if the image processing/object detection function worked while PiCar was in motion.

- More challenges: Improve FPS while live image processing and PiCar in motion

As expected, because of Pi's CPU processing limitations, we noticed a significant drop in FPS while PiCar was doing both image processing and navigating at the same time, which led to PiCar not stopping at the stop sign or the red traffic light. After some search on the Internet and reading articles, we'd come to the following solutions that also answer the question requested.

1. *Would hardware acceleration help in image processing? Have the packages mentioned above leveraged it? If not, how could you properly leverage hardware acceleration?*

   Referring to this discussion thread[5], to accelerate the hardware, one of them suggested bumping up the GPU memory to our PiCamera. So we increased the Pi GPU memory from the default size 128 MB to 512 MB. We then tried again running the program but the improvement wasn't as good as we expected. So we went back online and found this discussion[6] which the answer mentioned "overclocking". Intrigued by the phrase so we did some digging and found this article[7] was very useful as it explained overclocking in detail and how to do it. Therefore, after reading the materials, we tried overclocking Pi GPU frequency from 700 MHz to 1000 MHz.

   Result: Yes, hardware acceleration did have some help in improving FPS while doing image processing and navigating. But the result was unstable so we figured there must be something else we could do.

2. *Would multithreading help increase (or hurt) the performance of your program?*

---

[5] Reference: https://raspberrypi.stackexchange.com/questions/49574/getting-extremely-low-fps-on-rpi-3-while-image-processing

[6] Reference: https://raspberrypi.stackexchange.com/questions/57027/increase-webcam-fps-raspberry-pi-processing-speed-increase

[7] Reference: https://magpi.raspberrypi.com/articles/how-to-overclock-raspberry-pi-4

Inspired by this discussion thread[8], the next step we decided to also try adding Python multithreading programming to our script. Simply put, we opened up another thread for PiCamera to get camera feed and primitively process images, and the other would do more advanced image recognition and navigation. After adding multithreading, the improvement was noticeable and it increased the performance of our program.

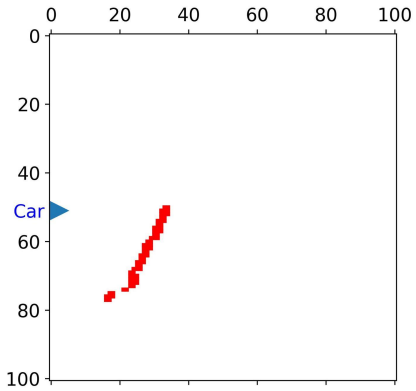
3. *How would you choose the trade-off between frame rate and detection accuracy?*

   Although the performance improved greatly after finishing the above steps, to put the cherry on top, we also tried reducing the camera feed image resolution from the original 640x480 to 300x300. The detection accuracy slightly dropped by approximately 3-5% but we thought it was acceptable and more importantly, PiCar reacted more timely and the program worked better than before.

Object detection consideration met:
[If the object detection API with tensorflow functions and the students demonstrate some ideas that they tried in order to speed up the fps, they get a 10.]

Step 7: Self-Driving Navigation (Routing)

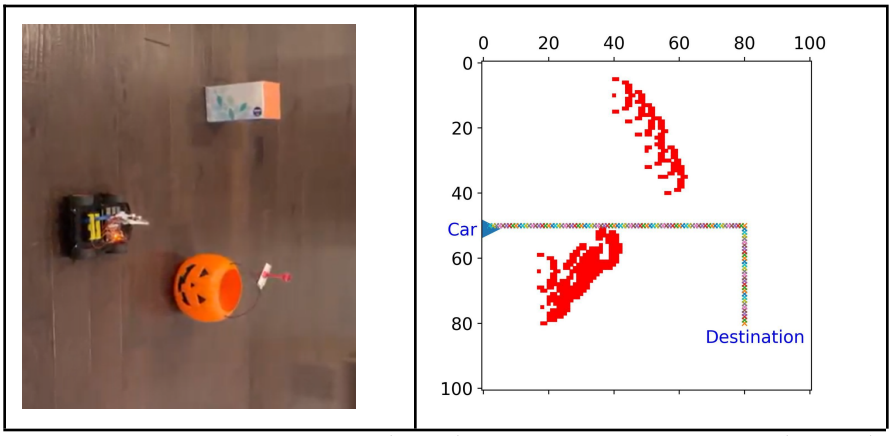
From step 5, a more advanced mapping grip can be generated. To navigate from a starting point to destination, a path finding algorithm needs to be developed. The map is a grid with either 0 or 1 at the positions. For example, the map below is a grid of 101 x 101 size with the car at position (51, 1). The goal of the path finding algorithm is to find the shortest path from the car's starting position to any destination in the grid.



Map grid generated by mapping

[8] Reference: https://raspberrypi.stackexchange.com/questions/57027/increase-webcam-fps-raspberry-pi-processing-speed-increase

The path finding problem is well studied in graph search problems. We used A* algorithm to find the optimal path from start position to destination[9]. A* algorithm is a variation on Dijkstra's algorithm. Dijkstra's algorithm starts from the starting point and expands outwards to neighbor points. It prioritizes the points with lowest total distance between the point and the starting point. A* algorithm uses an additional parameter to further optimize the speed of finding the shortest path between starting position and destination. In our case, the parameter is the distance between the current point and the destination. The sum of actual path distance between the starting point and the candidate point and the heuristic estimation of the distance between the candidate point and the destination, is used to prioritize the searching direction.



A* path: starting coordinate (51, 1), destination coordinate (80, 80)

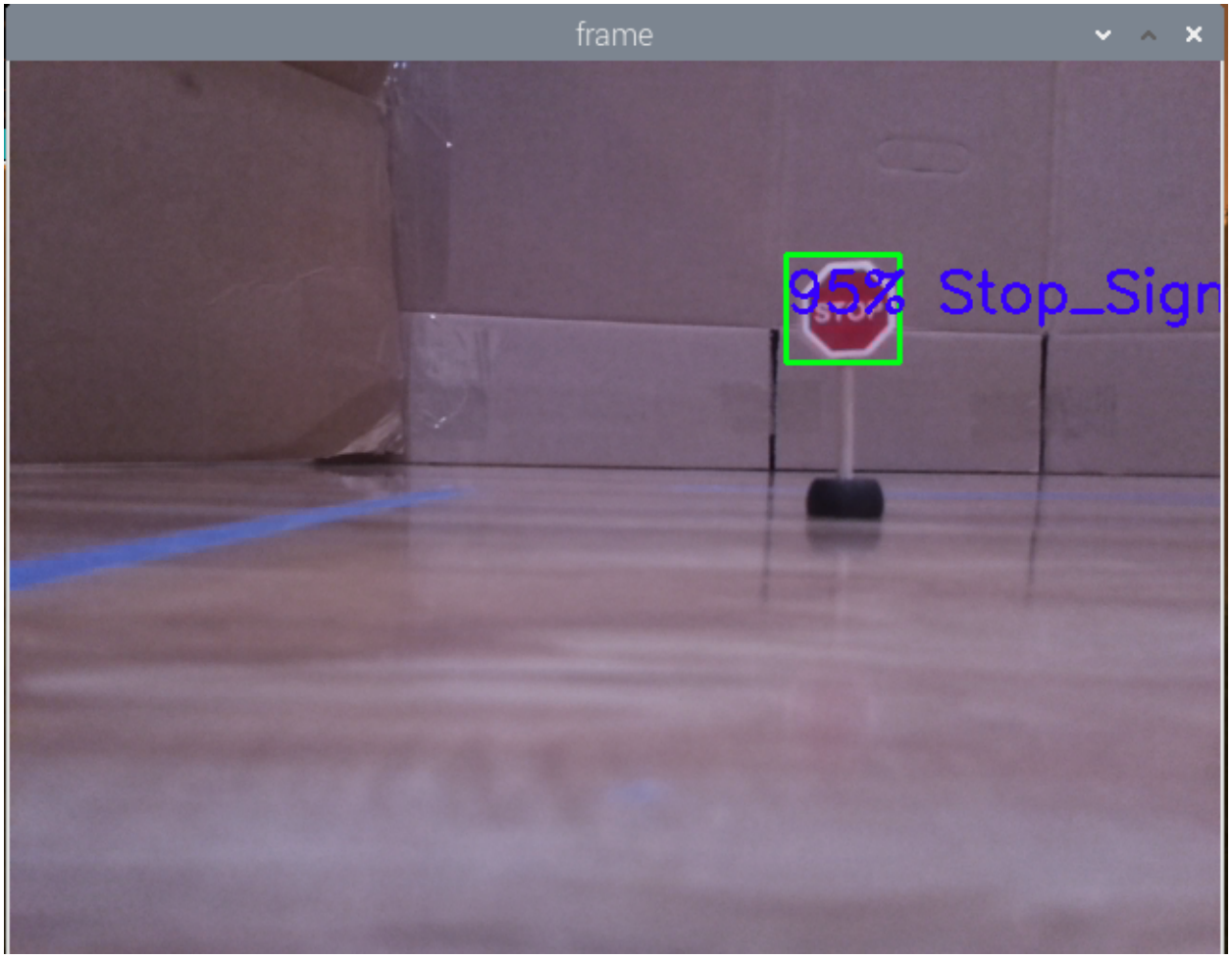
After finding the path, the car will move selected steps (we used 30) then perform mapping and path finding to update the map grid and optimum path.

One challenge is tracking the original destination after each movement. We used a variable to update the destination after the car moves to a new location. After every movement, a new destination is calculated based on the coordinates of previous location, current location and previous destination. This new destination is used in the path finding after obtaining an updated map grid. This process repeats until the car reaches the destination.

Another challenge is precisely controlling the car movement (direction and distance). After each movement, the new destination is calculated based on the coordinate of current location. If there is a large error in the movement control, the new destination will not be accurate and the car will target the wrong location. Due to the physical limitation of the car, we cannot achieve precise control using the current hardware (no GPS and gyroscope sensor are available), we calibrated the car angle and distance control with regression analysis by tuning the car power and the duration time. The final result is desirable and the car has decent movement precision.

[9] Ref: http://theory.stanford.edu/~amitp/GameProgramming

Testing



Traffic sign detection on PiCar: stop sign

```
pi@raspberrypi:~/picar-4wd/examples/CS437-IoT-Lab1 $ python3 test_multithreading
.py
Loading test/traffic_signs_recognition_edgetpu.tflite with test/dict.txt labels.
red traffic light detected.
stop sign detected.
```

Terminal message from traffic sign detection on PiCar

The first image above demonstrated that our PiCar can identify traffic signs/objects and also take actions while driving. The second image above from our terminal demonstrated that our PiCar actually "saw" the sign and identified it correctly.

More Challenges:

1. In the beginning, it's much easier to detect our stop sign than a red traffic light with our model trained with an insufficient amount of images, and the camera

resolution also had an impact on smaller objects such as our red traffic light. The issue was resolved later with more images provided to our trained model.

2. The performance of our PiCar has significant relevance with our hardware parts, including, the tire resistance to the ground, the camera resolution, the capability of the chips on the Raspberry Pi, and the powers of the four motors driving our tires, the ground materials of the running course, which is hard to be controlled. The problems were resolved with numerous trials and calibrations.

The YouTube video provided below contains two testing parts with two different paths and different traffic signs/objects respectively. The course reviews are provided at the beginning of each test to display a clearer sight about the course that our PiCar is about to run through.

In the first part, our PiCar was planned to go through a first green traffic light, a first stop sign, a second green traffic light, a second stop sign, a third stop sign, and finally a 7 cm height toy character from Star War. When the car went through the first green traffic light, it didn't stop, and then it stopped at the first stop sign, following with a right turn. The PiCar further went through a second green traffic light without stopping, and then stopped and turned right at the second stop sign. It further stopped and turned right again at the third stop sign toward the toy character. It then finally stopped in front of the character while seeing it.

In the second part, our PiCar was planned to go through a first green traffic light, a stop sign, a second green traffic light, and finally a red traffic light. In the second part, the PiCar did similar activities to those in testing part one, including passing two green traffic lights without stopping and also stopping and turning at stop signs. It finally stopped in front of the red traffic light when seeing it. The second part demonstrated the solid performance and repeatability of our PiCar.

YouTube links for PiCar test runs:
https://youtu.be/ASUa1TF8Fws
This demo video is for Lab 1 Part 2 Step 8, which shows exactly what is explained in **Design considerations for Part 2 Step 8.**

Schematics of test courses. (above) course for test 1, (bottom) course for test 2.