P2M PROJECT:

SUP'COM
Ecole Supérieure des Communications de Tunis

1. Documentation

Intel Neural Compute Stick2 (NCS2) :

NCS2 acts as a USB accelerator

Intel Movidius Myriad X Vision Processing Unit (VPU) a specialized processors for machine learning

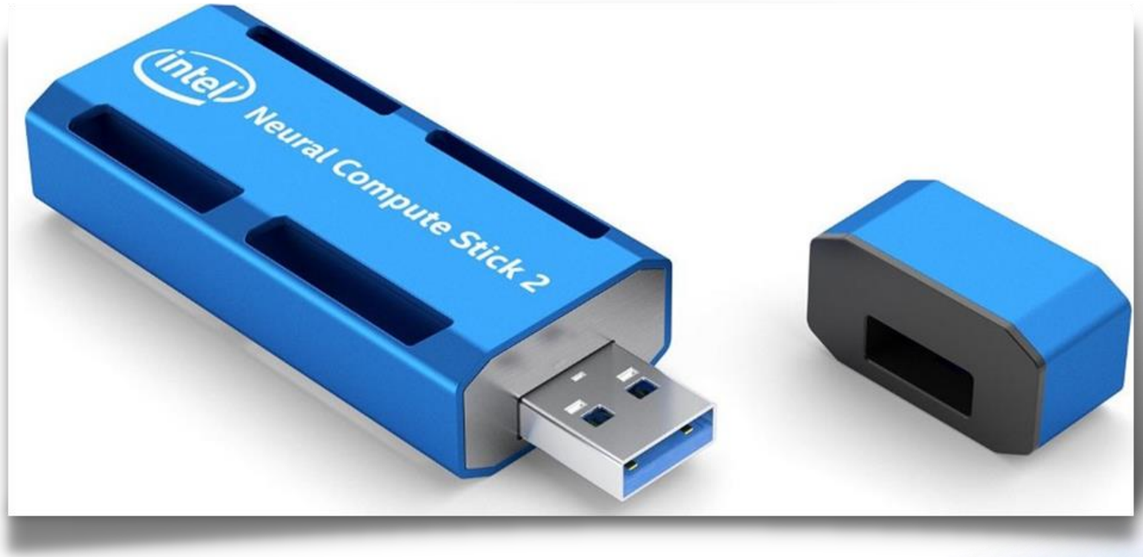
---

Accelerates TensorFlow neural network inferences and improves performances by 10x factor

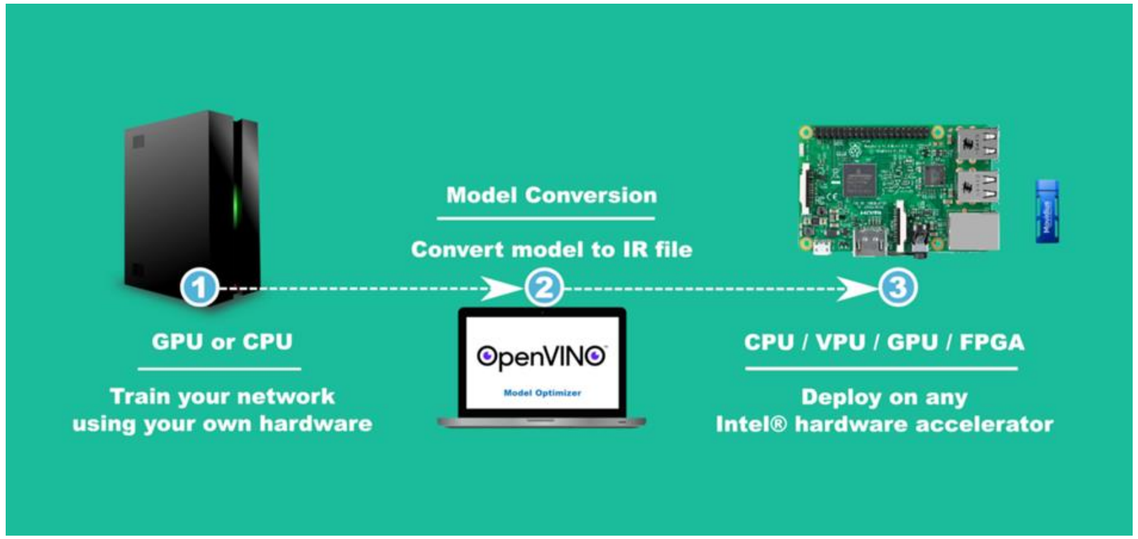
---

Supports multiple softwares (Ubuntu, CentOS, Windows 10...) and multiple algorithms (TensorFlow, Caffe, ApacheMXNet...) via an Open Neural Network Exchange conversion.

Used in our project to achieve end to end learning for self-driving car + traffic sign and pedestrian detection and handling object detection models with deep learning D-CNN (mobilenetv2ssdcoco , mobilenetv2ssdcoco quantized...)



How to connect the NCS2 to the RPi :

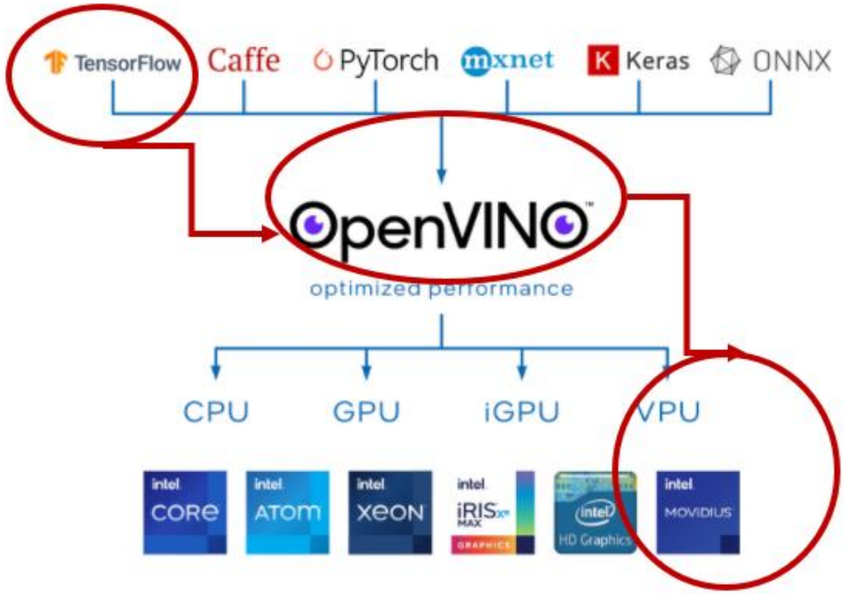


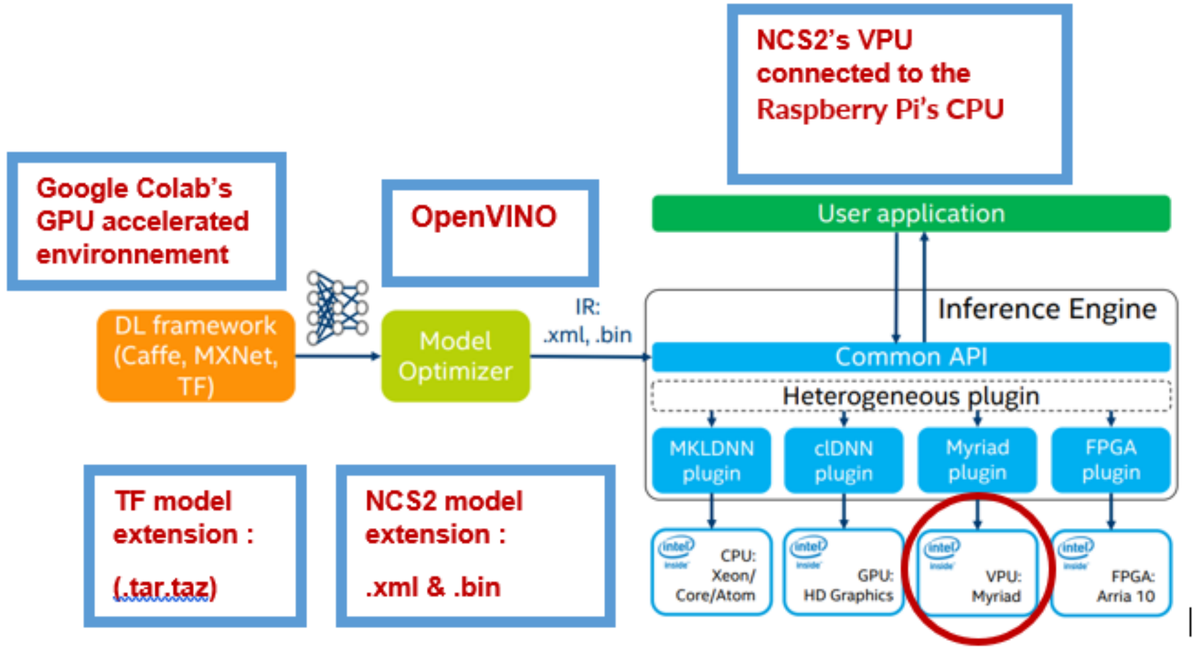
To connect the NCS2 to the RPi, the procedure required is:

1. Convert a TensorFlow model to NCS compatible one, using Open VINO Toolkit by Intel (all of this work is done on your pc's CPU or GPU, for TensorFlow's model we use Google Colab)

2. Install a light version of Open VINO on Raspberry, to run inferences onboard

3. Test and deploy the converted model on Raspberry





SUP'COM
Ecole Supérieure des Communications de Tunis

2.Testing the NCS2

2.1 Demo_Security_Barrier_Camera :

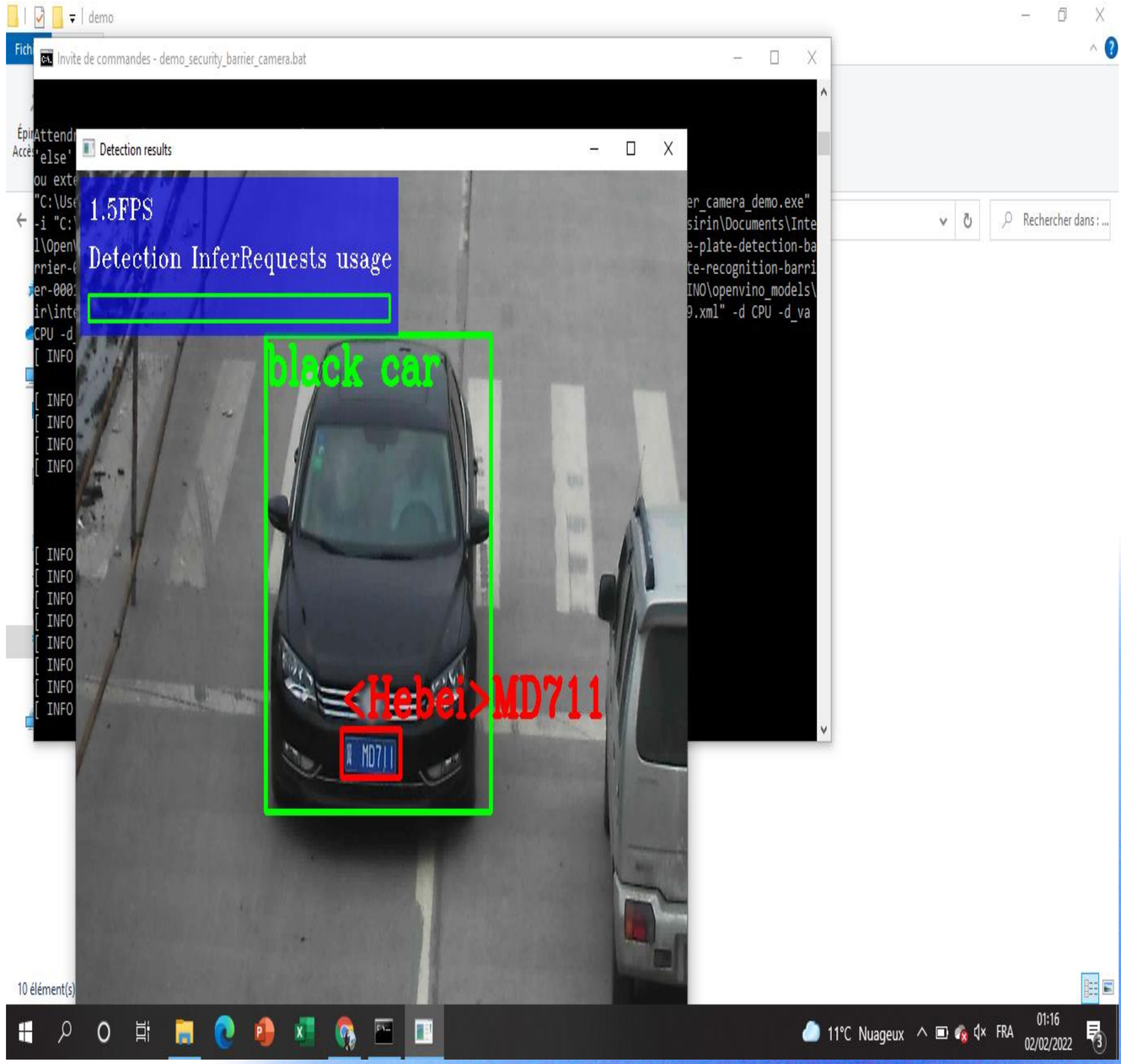
Using the deplyement tools offered by openvino

It uses SqueezeNet for the Computer Vision and Pattern Recognition (object detection : balck car and the licence numbers)

The inference is run in the VPU of the neural compute stick (we must specify the command line -d MYRIAD when we run the Demo)

Software (my host machine Windows 10 with a lite version of intel openvino_2021.4.752 installed and configured,Squeezenet DNN) Hardware(NCS plugged in my PC’s USB port).

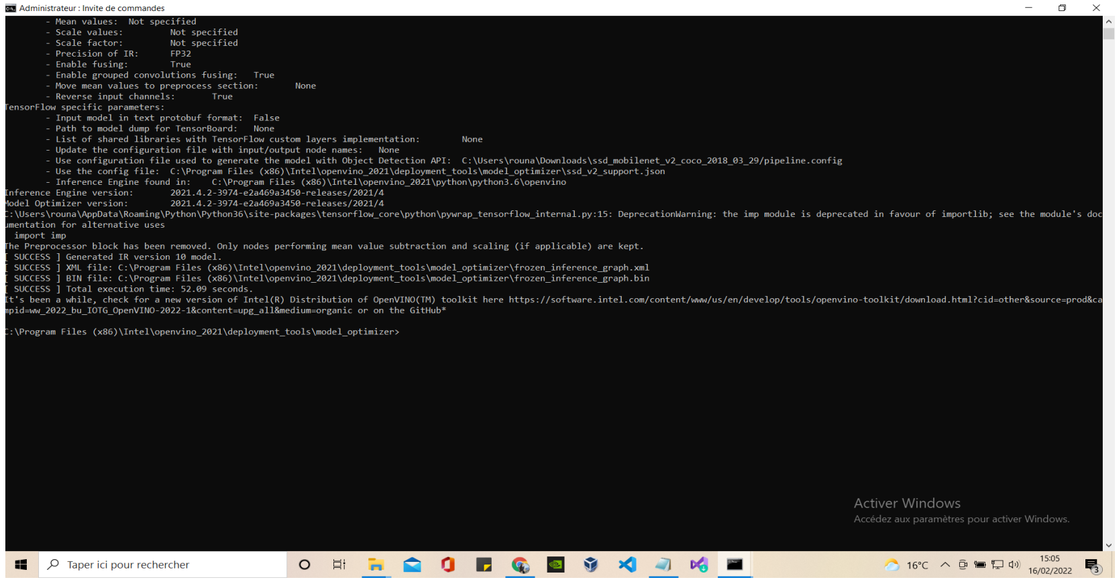
This Demo downloads a SqueezeNet model, uses the Model Optimizer to convert the model to the .bin and .xml Intermediate Representation (IR) files.The Inference Engine requires this model conversion so it can use the IR as input and achieve optimum performance on Intel hardware.



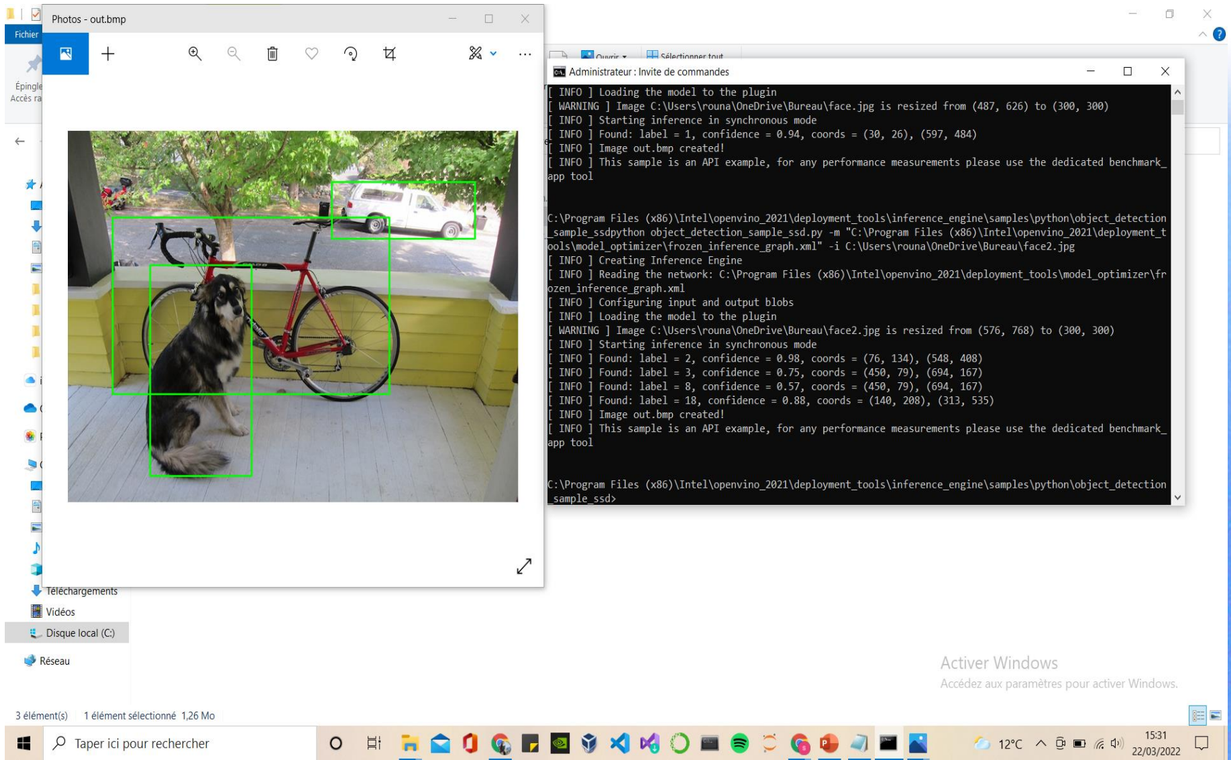
2.2 Demo with ssd_mobilenet_v2_coco_2018_03_29 on images :

First we download the pretrained model ssd_mobilenet_v2_coco from model ZOO

Then we use Openvino's model optimizer to do the IR convertion of the model and generate from it's frozen_inference_grapg.pb both XML file(frozen_inference_grapg.xml) and BIN file (frozen_inference_grapg.bin)



Then we run the XML file using Openvino's inference engine That uses object_detection_sample_ssd.py offred along in the python samples of Openvino



The output of this Demo is the input image with a boundary box showing the face detection.

Software (my host machine Windows 10 with a lite version of intel openvino_2021.4.752 installed and configured,ssd_mobilenet_v2_coco_2012_03_29 DNN) Hardware(NCS plugged in my PC's USB port).

2.3 Demo with ssd_mobilenet_v2_coco_2018_03_29 on videos for real-time object detection :

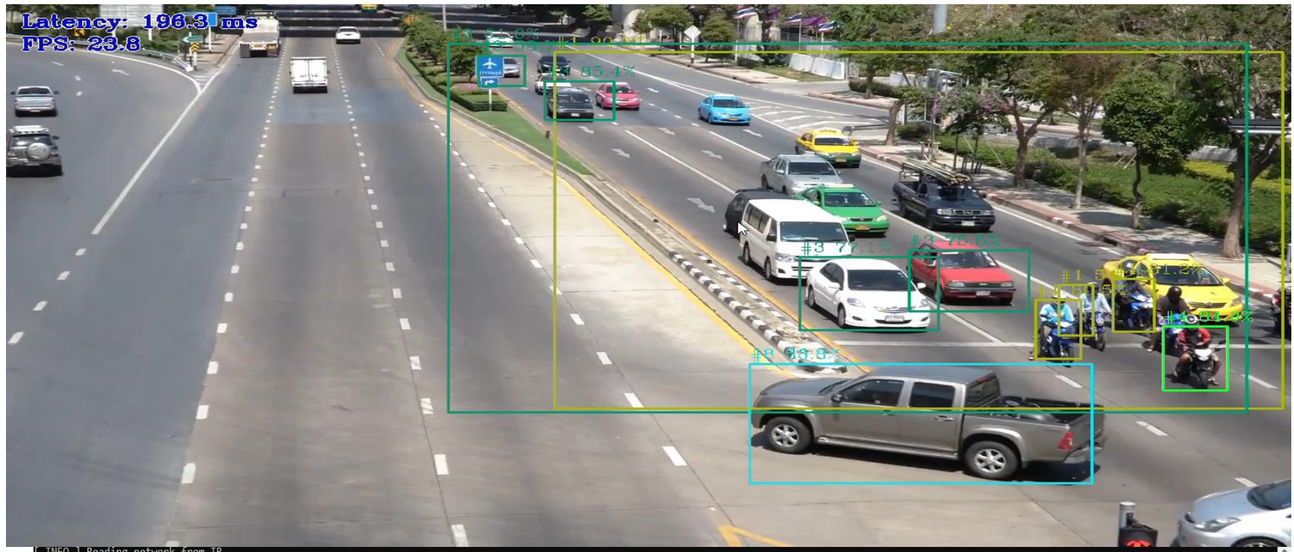
First we download the pretrained model ssd_mobilenet_v2_coco from model ZOO

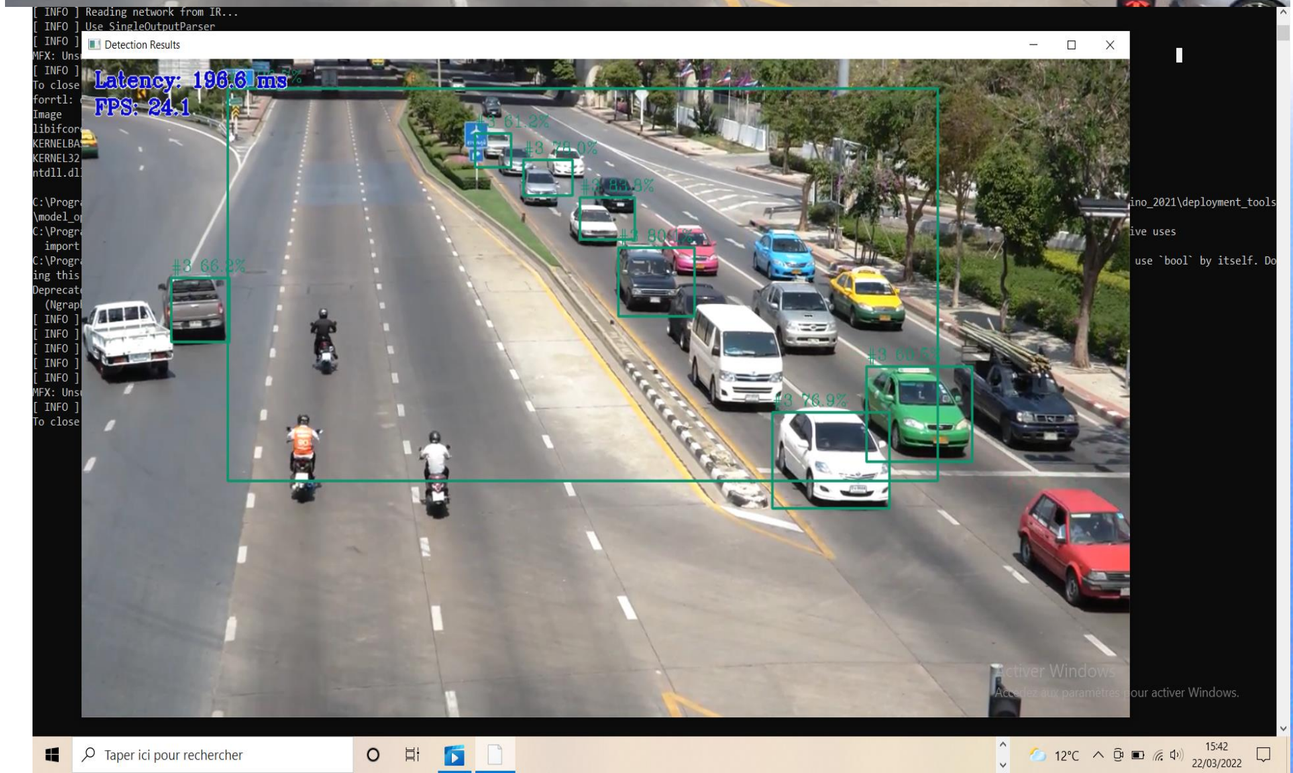
Then we use Openvino's model optimizer to do the IR convertion of the model and generate from it's frozen_inference_grapg.pb both XML file(frozen_inference_grapg.xml) and BIN file (frozen_inference_grapg.bin)

Then we run the XML file using Openvino's inference engine That uses object_detection_demo.py offred along in the open_model_zoo of Openvino's deployement tools.

The output of this Demo is the input video with a real time object detection boundary boxes.

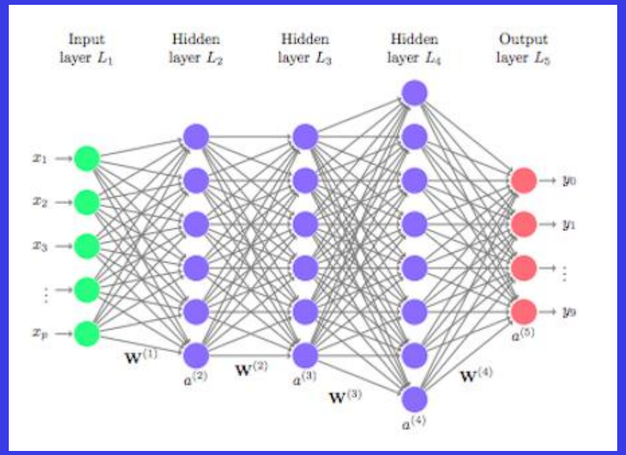
Software (my host machine Windows 10 with a lite version of intel openvino_2021.4.752 installed and configured,ssd_mobilenet_v2_coco_2012_03_29 DNN) Hardware(NCS plugged in my PC's USB port).





3. Lane Following:

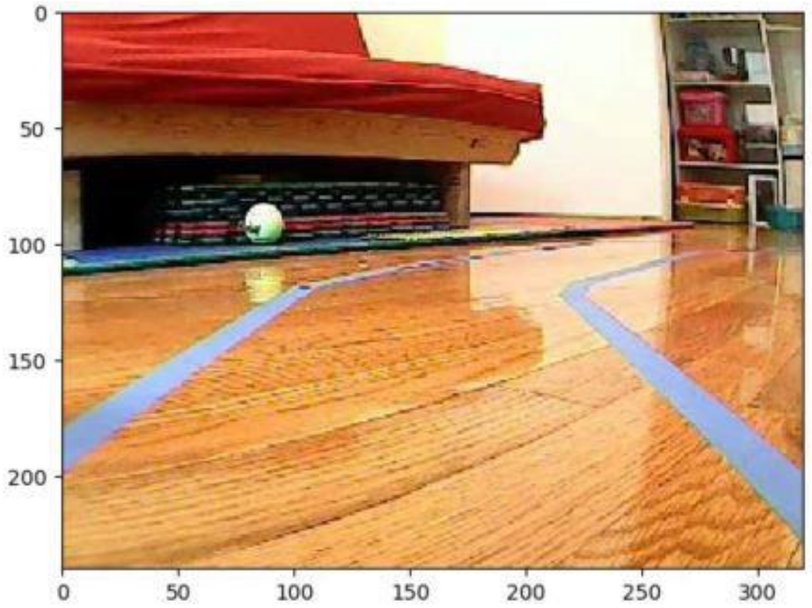
- Lane following using Open CV

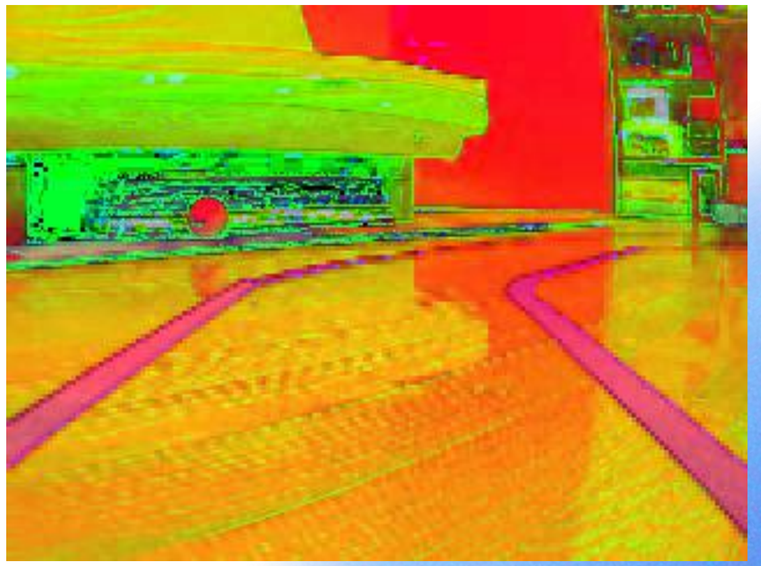


- Lane following using deep learning

Using only Open CV

1) Isolating the Color of the Lane





2) Create a mask for the lanes

```
1   lower_blue = np.array([60, 40, 40])
2   upper_blue = np.array([150, 255, 255])
3   mask = cv2.inRange(hsv, lower_blue, upper_blue)
```

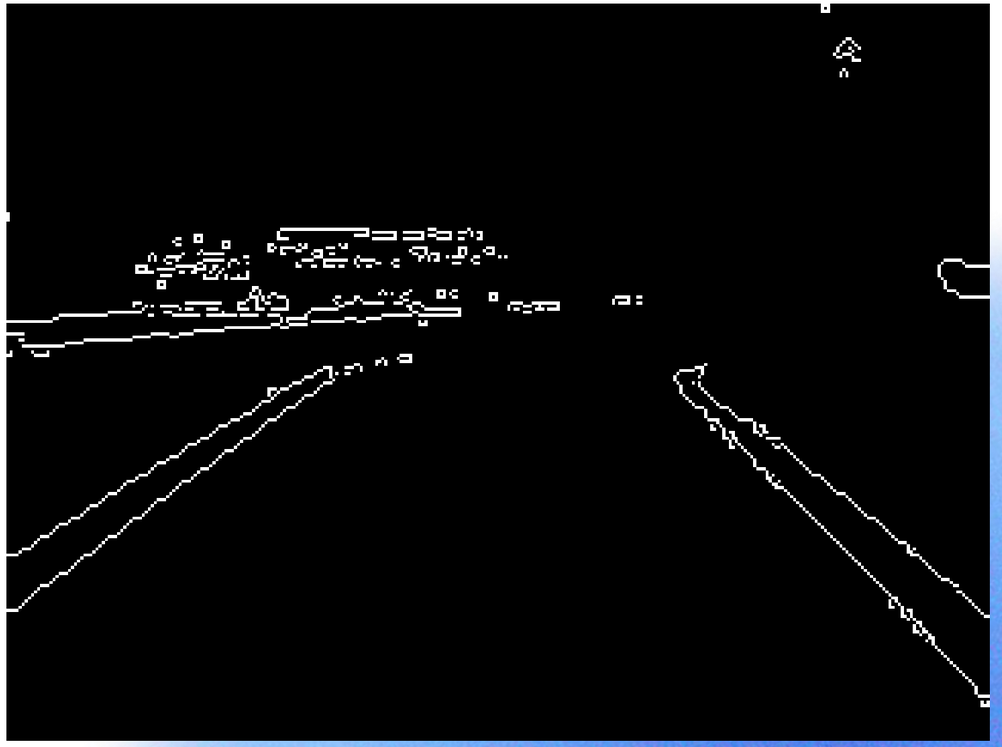
blue_color_mask.py hosted with ❤ by GitHub

3) Detecting Edges of Lane Lines

Using the Canny edge detection function

```
1    edges = cv2.Canny(mask, 200, 400)
```

canny_edge_detection.py hosted with ❤ by GitHub



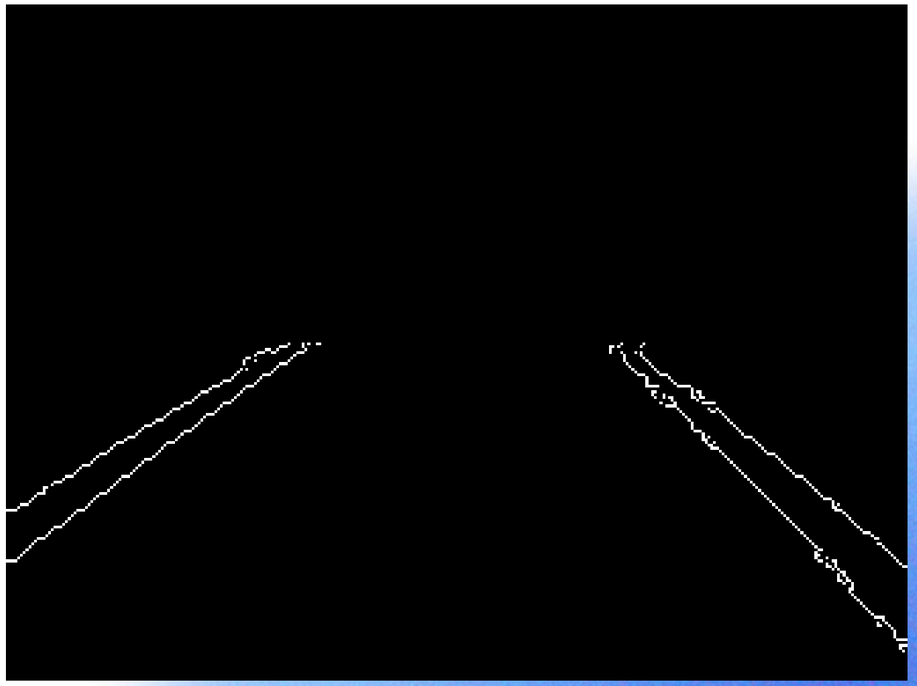
4) Isolate Region of Interest

Removing the top part of the edges image

```python
def region_of_interest(edges):
    height, width = edges.shape
    mask = np.zeros_like(edges)

    # only focus bottom half of the screen
    polygon = np.array([[
        (0, height * 1 / 2),
        (width, height * 1 / 2),
        (width, height),
        (0, height),
    ]], np.int32)

    cv2.fillPoly(mask, polygon, 255)
    cropped_edges = cv2.bitwise_and(edges, mask)
    return cropped_edges
```



PiCar

5) Detect Line Segments

Using the Hough transform function

[9]:
```python
#Detect Line Segments

def detect_line_segments(cropped_edges):
    # tuning min_threshold, minLineLength, maxLineGap is a trial and error process by hand
    rho = 1  # distance precision in pixel, i.e. 1 pixel
    angle = np.pi / 180  # angular precision in radian, i.e. 1 degree
    min_threshold = 10  # minimal of votes
    line_segments = cv2.HoughLinesP(cropped_edges, rho, angle, min_threshold,
                                    np.array([]), minLineLength=8, maxLineGap=4)

    return line_segments

line_segments = detect_line_segments(cropped_edges)
print(line_segments)
```

(x1 , y1)	(x2 , y2)
[[[0 180	101 120]]
[[223 124	236 134]]
[[250 143	318 198]]
[[0 197	55 157]]
[[219 124	312 217]]

6) Combine Line Segments into Two Lane Lines

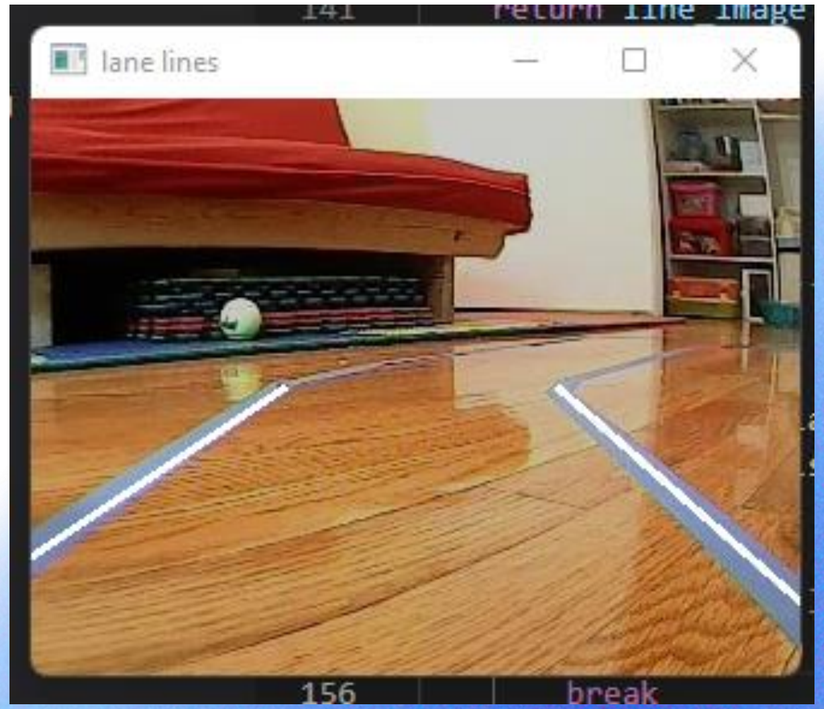
Combine the identified line segments into Two Lane Lines using the direction of their slope.

In the end we get the coordinates of the endpoints of the two lanes we need.

```
lane_lines = average_slope_intercept(frame, line_segments)

print(lane_lines)
```

```
[[[-75, 240, 105, 120]], [[355, 240, 219, 120]]]
```



Special Cases :

One lane line in the image:

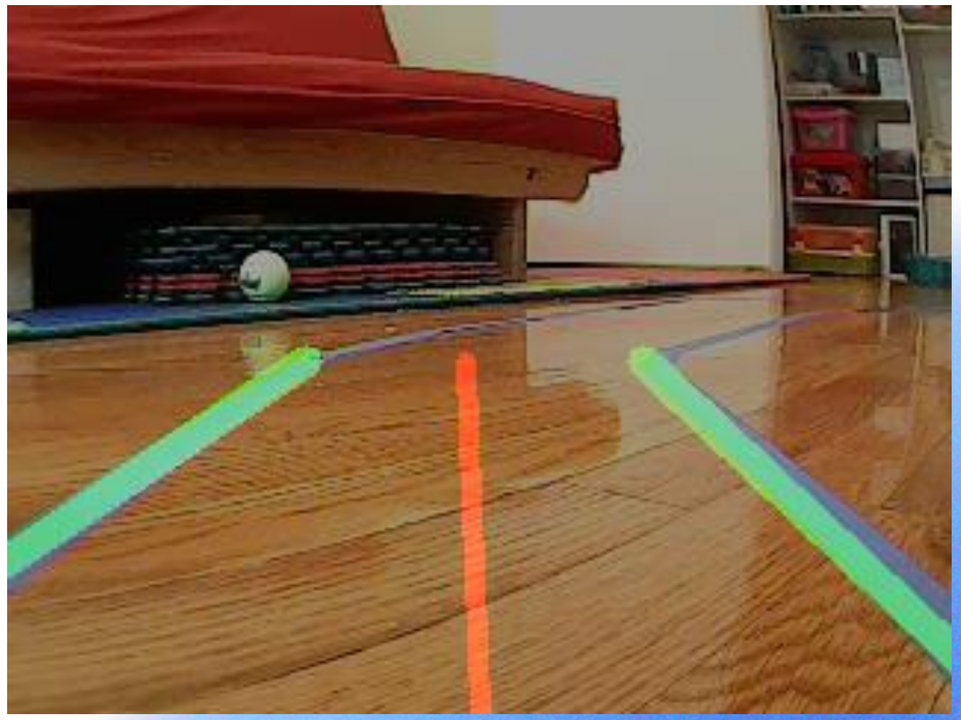
Vertical line segments:

PiCar

Steering

1) If we detect 2 lanes

- Calculate the average of the top coordinates of the lanes.
- Calculate the middle of the image,
- Draw a line from the middle of the bottom of the image to the average calculated
- Give the steering the angle that the line is forming with the X axis

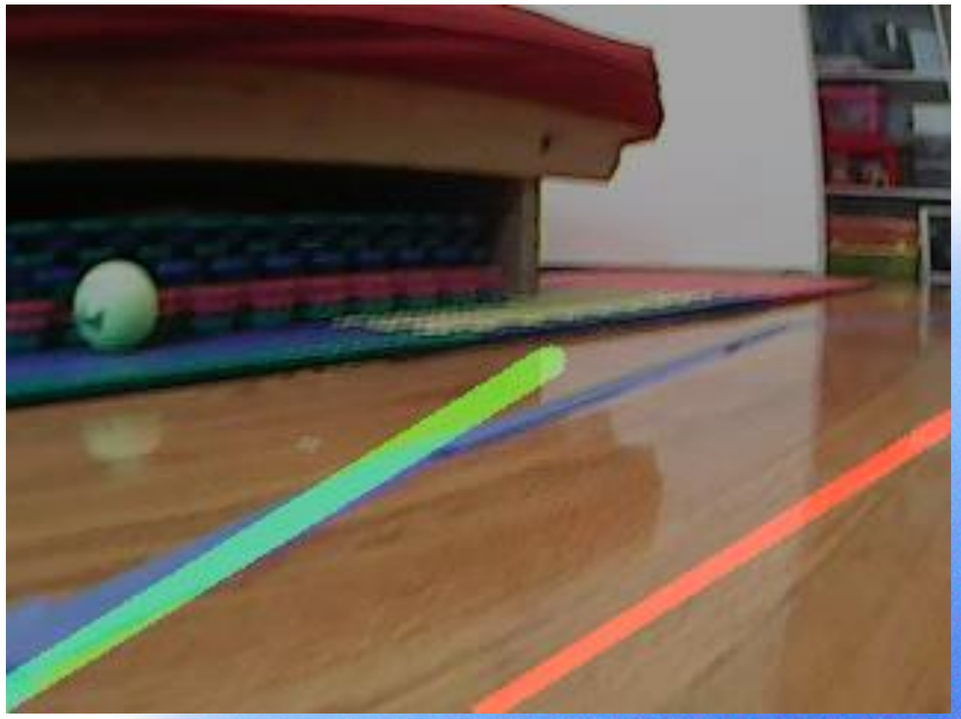


PiCar

Steering

2) If we detect 1 lane

- Draw a line parallel to the lane passing through the middle of the bottom of the image
- Give the steering the angle that the line is forming with the X axis



Calculating the steering angle

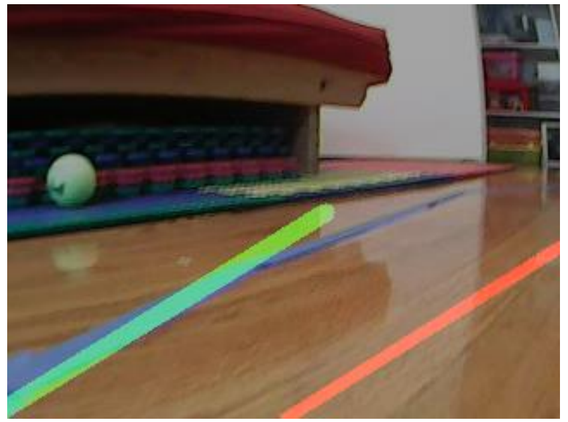
For this PiCar, the steering angle of :

- 90 degrees ➔ heading straight,
- 45–89 degrees ➔ turning left,
- 91–135 degrees ➔ turning right.

```
angle_to_mid_radian = math.atan(x_offset / y_offset)
angle_to_mid_deg = int(angle_to_mid_radian * 180.0 / math.pi)
steering_angle = angle_to_mid_deg + 90
```

PiCar

Stabilisation



Some times, the steering angle may be around 90 degrees (heading straight) for a while, but, for whatever reason, the computed steering angle could suddenly jump wildly, to say 120 (sharp right) or 70 degrees(sharp left).

To smoothen up the steering : if the new angle is more than **max_angle_deviation** degree from the current angle, just steer up to **max_angle_deviation**

P2M PROJECT:

SUP'COM
Ecole Supérieure des Communications de Tunis

3. End-to-End Lane Navigation via Nvidia’s Deep Learning Model (using Colab’s GPUs)

SUP'COM
Ecole Supérieure des Communications de Tunis

4.Next Steps

Convert our Deep Learning model to the .bin and .xml Intermediate Representation (IR) files to run them over the NCS2 and RPI3 :

After generation the deep learning model for autonomous lane navigation (Lane_navigation_finale.h5 and Lane_navigation_check.h5)

We will use intel's **DL Workbench** that converts Keras H5 models to the Saved Model format and then to the OpenVINO™ format (XML and BIN files) with the Model Optimizer.

The inference can then be run in the VPU of the neural compute stick and the Raspbarry Pi3

DL Workbench combines OpenVINO™ tools to assist you with the most commonly used tasks: import a model, analyze its performance and accuracy, visualize the outputs, optimize and prepare the model for deployment in a matter of minutes.