Hochschule
Ravensburg-Weingarten
University of Applied Sciences

Electrical Engineering and Embedded Systems

Embedded Project Report

Traffic Sign Detection using Intel Movidius Stick and Raspberry Pi on a pre-trained network under Tensorflow/Caffe Model

Project Team Details:

1. Samudra Gupta, 31647
2. Kishan Kumar Mandal, 31658

CONTENTS

INTRODUCTION

The Project is done in two parts:

Part 1: The first requirement is to use the Intel Neural Compute Movidius Stick (NCS) with RaspberryPi to learn how to utilize it as an edge computing platform and improve the inference speed and frame rate (FPS) of the Object detector. A pre-trained Caffe model trained on 21 classes viz. car, person, bus, etc. is deployed which uses the NCS as a Visual Processing Unit (VPU) to give a faster inference. The object detector detects with good accuracy with fps ~(7-8). Since no appropriate pre-trained object detector network under Caffe model was available that explicitly detects traffic signs, the reference to another deep learning framework such as **Tensorflow** is done.

Part 2: The focus is to detect Traffic sign using Tensorflow framework and deploying it on the embedded platform of RaspberryPi using ROS environment. This package when launched runs the object detector network, trained on 78 different classes of traffic signs and can detect them at a frame rate of ~0.4 fps on Raspberry Pi3 without the NCS.

PURPOSE

In the area of autonomous driving, the traffic sign is a crucial part to detect and act accordingly. There is a strong need to find a network which can both detect and classify traffic sign and could be deployed on an embedded platform such as RaspberryPi. Due to the lack of adequate computing and graphical processing power on Raspberry Pi, an additional VPU such as the Intel Movidius Neural Compute stick can significantly improve performance and increase the inference speed of the system. Further a Robotic Operating System (ROS) environment can be used to organize different functionality of an Autonomous car and improve the communication between different functional modules.

GOAL

Implementation of a Traffic sign detector using a pre-trained network under Tensorflow architecture and implementing the overall system on ROS thereby publishing the results of detection i.e., image ID, label, bounding box coordinates and the accuracy into a topic. The final goal is the usage of the traffic sign detector as a modular function in an autonomous RC car to work autonomously with other ADAS modules.

SYSTEM OVERVIEW

Project Part 1

Running a pre-trained object detection Caffe Model onto RaspberryPi using Intel Neural Compute Stick

1. Generating a graph file from a pre-trained Caffe model and loading the graph into the RaspberryPi for inferencing.
2. Calibrating the Fish-eye PiCamera to undistort the image for proper detection. Images of the distorted checkerboard are taken from the PiCamera which are used to finally calculate the parameters for the final output which gives undistorted images.

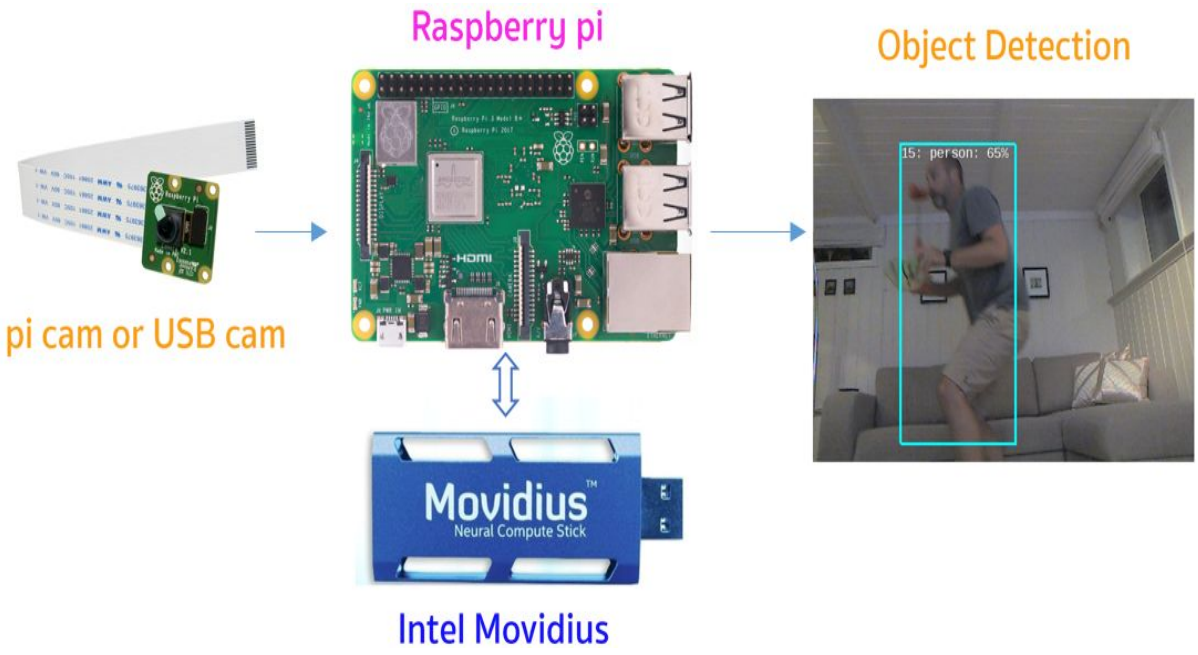

Fig. 1. Flow Chart describing process flow of Object Detection

Project Part 2

Running a pre-trained "Traffic Sign Detector" network under Tensorflow without the Neural Compute Stick.

1. Running a pre-trained traffic sign detector network under Tensorflow model in the RaspberryPi ROS environment.
2. To perform real-time inference on Live video and validate the traffic sign detector.
3. Publishing the results which include image ID, image label, the accuracy of detection and the bounding box coordinates, in a topic.

HARDWARE DESIGN

Camera Specifications

1. Raspberry Pi Camera w/fish-eye lens
2. Resolution: 5 Megapixel • Viewing Angle Degrees: 110°/160° • Frame-Rate: 1080p@30fps/720p@60fps

Raspberry Pi 3B+

1. Extended 40 pin GPIO header
2. Full-size HDMI
3. 4 USB 2.0 ports
4. 5V/2.5A DC power input
5. 1GB LPDDR2 SDRAM
6. Broadcom BCM2837B0, Cortex A53 (ARMv8) 64 bit SoC @ 1.4GHz
7. 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
8. Gigabit Ethernet over USB 2.0 (maximum throughput of 300 Mbps)
9. CSI camera port for connecting a Raspberry Pi camera
10. Micro SD port for loading your operating system and storing data
11. Power over Ethernet PoE) support (requires separate PoE HAT)

System CPU requirements

1. Ubuntu 16.04 or higher Version.
2. At least 8GB of RAM with dual-core CPU
3. At least 20GB of free storage
4. USB3 (preferred)

SOFTWARE DESIGN

1. The complete system is based on Raspian OS with ROS package already installed with important libraries like OpenCV.
2. Since the major objective of the project is to integrate it into an Autonomous RC-car with other ADAS functionality. The complete design is built keeping that in mind.
3. The embedded operating system is based on Raspbian Stretch with desktop, the Raspbian based on Debian Stretch. ROS Kinetic Kame and dependency software. This ready to use system image can be downloaded from [8]
4. A ROS package is created which when launched, can perform the detection task and publish results to the topic Object detection node using appropriate message.
5. Any subscriber can subscribe to the topic and use the results for other functionality.

RESULTS AND DISCUSSION

Project Part 1

1. **Prerequisites**: Following the guidelines mentioned in the below link, we will set up the environment. [1]
2. Download the Repository of the pre-trained object detection network under Caffe model, with the required files.
3. Plug in the Movidius stick and generate the graph file.[2]
4. We generate the graph file on the development PC using two files namely: A .**prototxt** file which defines the model architecture and a **"weights"** file with all the layer weights.
5. The following command is run inside the directory, to compile and generate the graph file and import it onto the Movidius Stick: ***mvNCCompile *.prototxt -w *.caffemodel -s 12 -o graph/graph***
6. Calibration of Fish-eye PiCamera is done and the reference website is used for the necessary code and concepts.[7].
7. The necessary changes in the main program file are made to undistort the video feed and obtain good performance results.
8. When the package is run it can detect objects belonging to 21 different classes having good accuracy with fps ~(7-8).

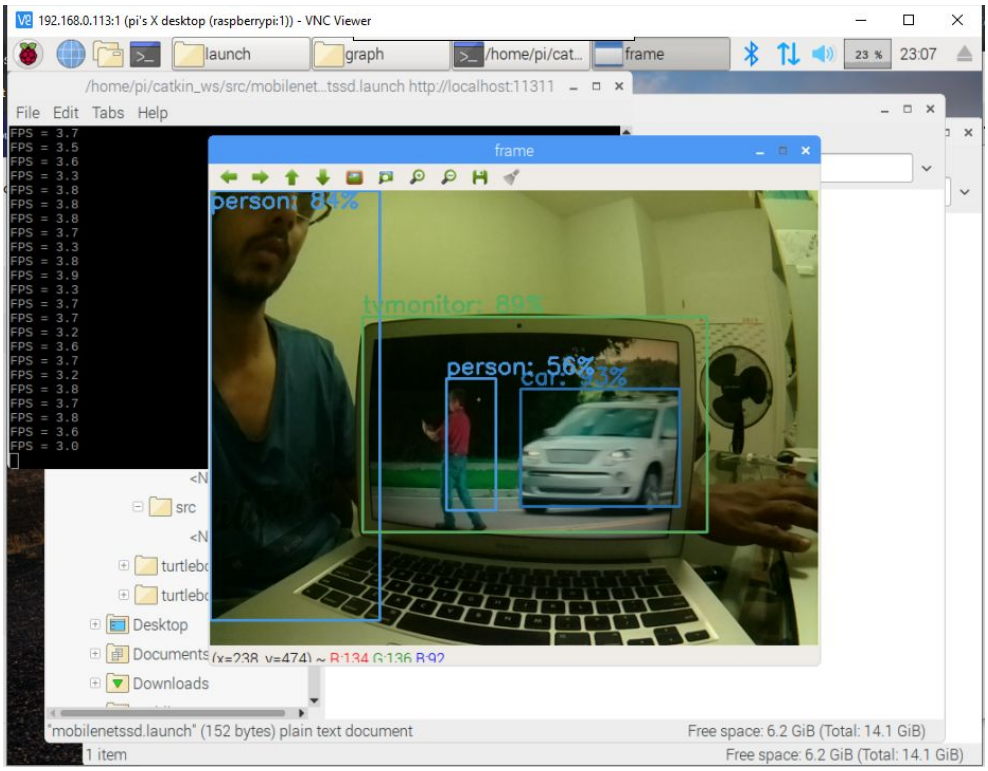

Fig. 2. Object Detection With Image Calibration implemented. Reduced FPS by a factor of ~2

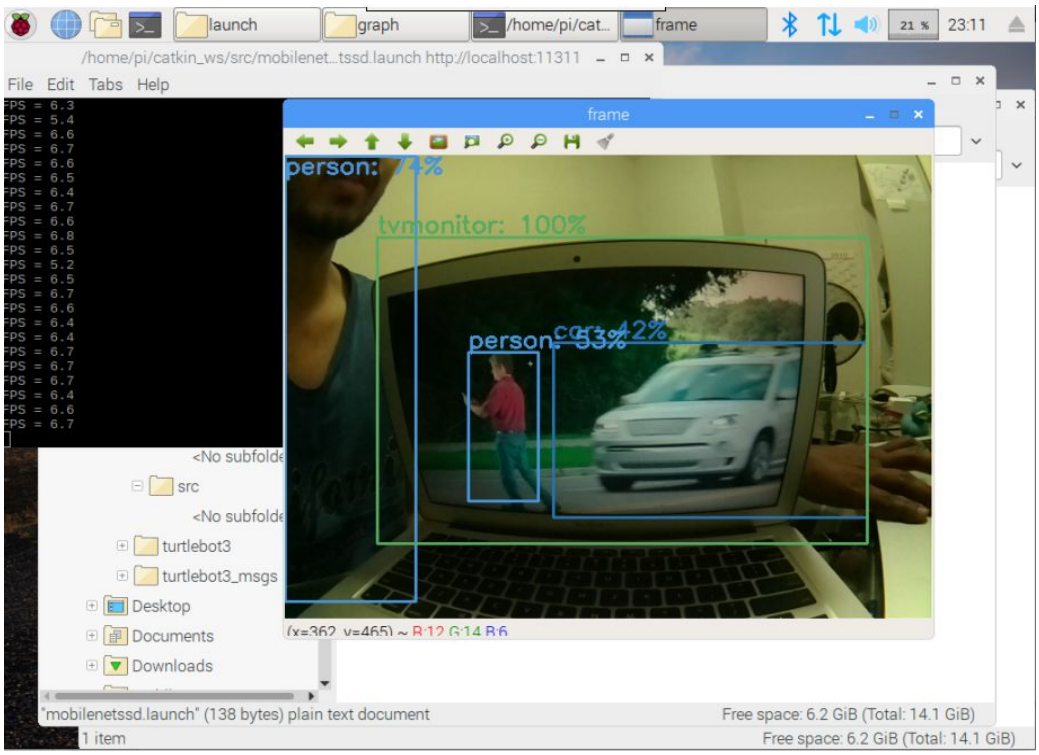

Fig. 3. Object Detection without Image Calibration implemented.

Project Part 2

1. The very first task is to set up the Tensorflow environment on the RaspberryPi which is a bit complicated and confusing task at the beginning. After much research and trail, the best way to set up could be referred from this website[9], until the step of installing Tensorflow.
2. Since OpenCV is already available as a ROS package, OpenCV is not installed again.
3. The same above link downloads a pre-trained object detection model and it is suggested to download a lite model (ssdlite_mobilenet_v2_coco_2018_05_09).
4. To test it using the RaspberryPI, the setup was done according to the Github repository [10] under the heading “Detect Objects.”
5. A pre-trained traffic sign detection model is downloaded [11]. The model successfully detects traffic signs belonging to 78 different classes.
6. A ROS package and a new message (traffic_sign.msg) is created. This Traffic Sign Object Detector message comprises Class name, Image-ID, Bounding-Box coordinates and the percentage accuracy of detection (score).
7. The live inference is made and results can be seen in the pictures below. Majority of the traffic signs are detected and labeled correctly.

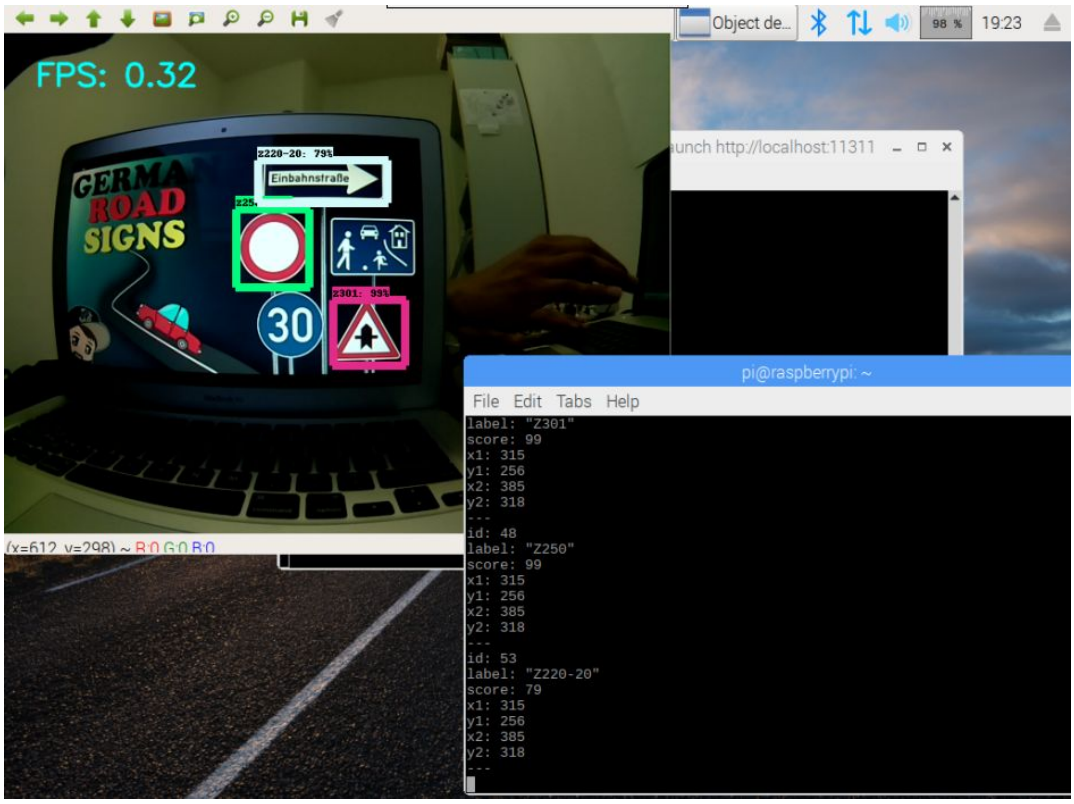

Fig.4. Detection of German Road Signs.

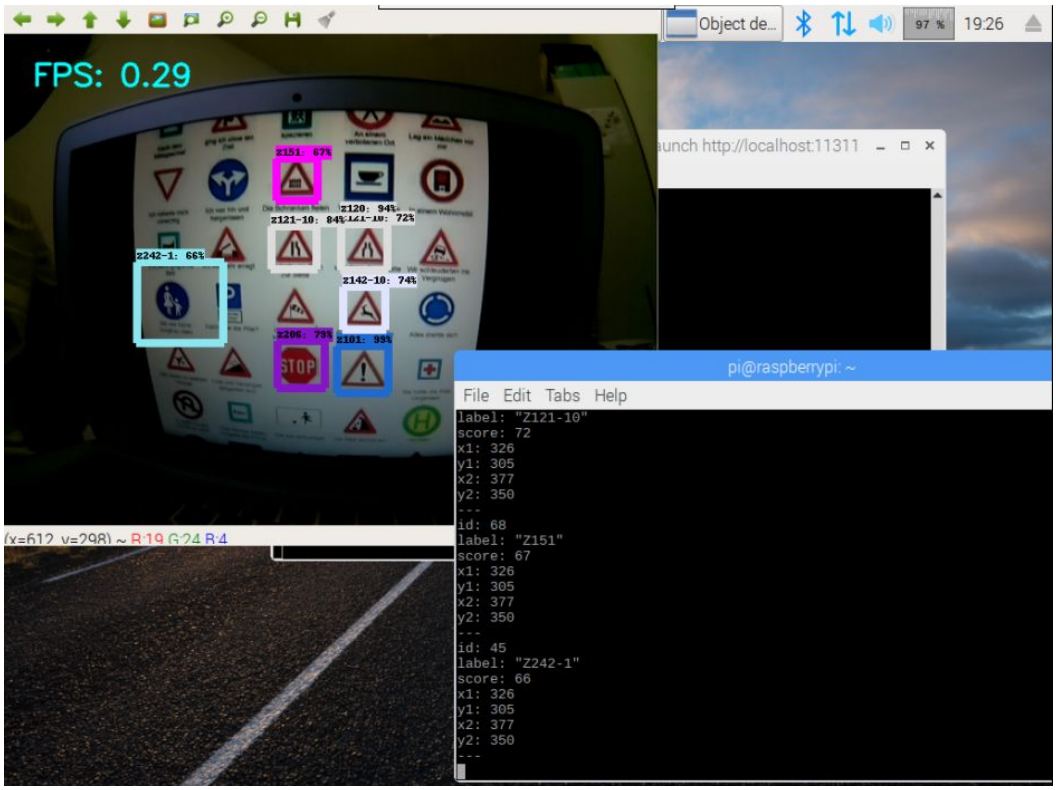

Fig. 5. Detection of German Traffic signs.

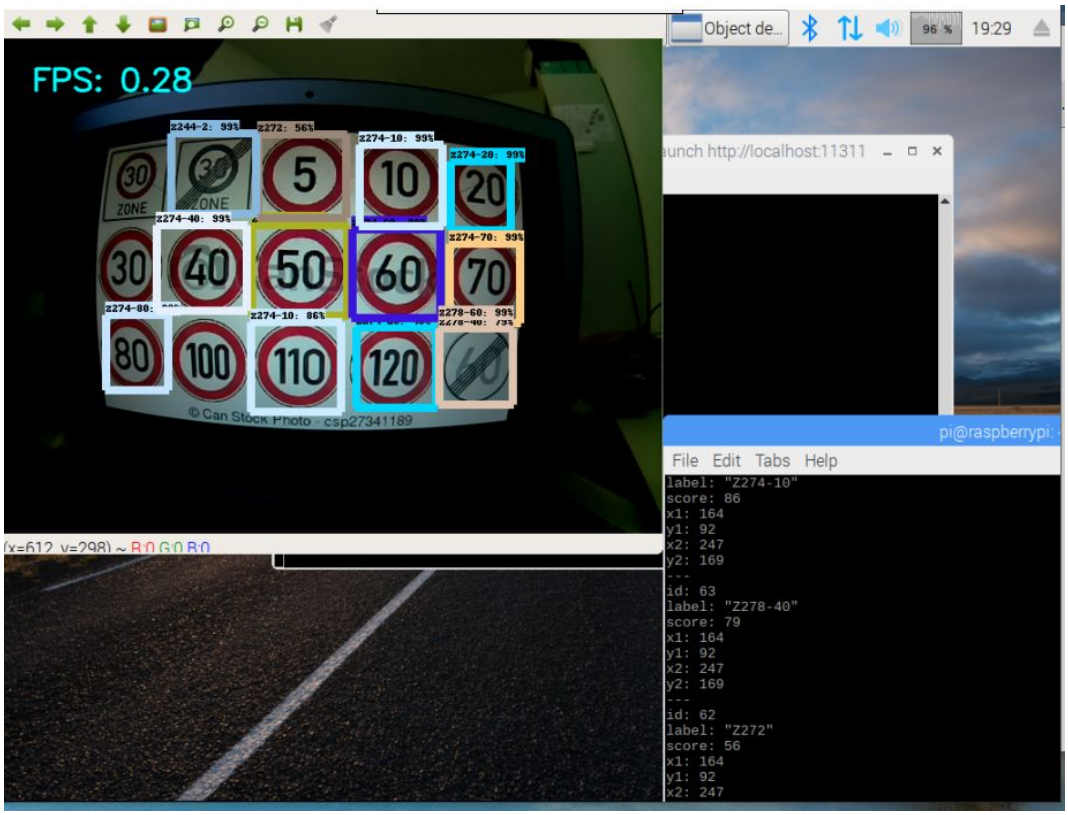

Fig. 6. Detection of different speed limit category.

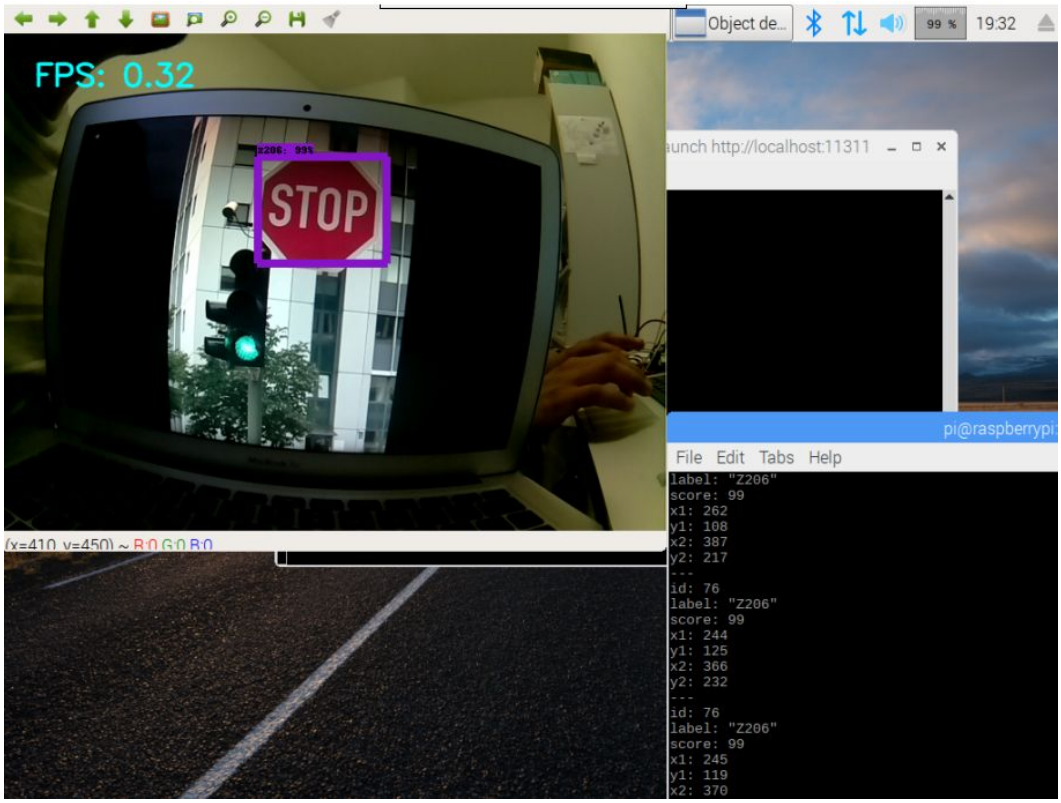

Fig. 7. Detection of German Road Signs.- STOP Signal.

CONCLUSION

1. We have successfully generated the graph file out of the pre-trained Caffe model and ported it onto the Intel Movidius stick for faster inference and higher FPS.
2. Also, we have successfully run a pre-trained network under Tensorflow model on ROS environment but whose FPS is quite slow.(~0.4 FPS)
3. Using a fish-eye camera generates distorted images. So, using a script to remove this distortion helps in more accurate inferences.
4. Also, the Traffic Sign detector detects most of the traffic sign successfully.
5. The GitHub repository link provided[11] has only the *frozen_inference_graph .pb* and *label_map.txt* files. According to the link[12], the *checkpoint* and *meta* files are also required to successfully port the Tensorflow model into a Graph file onto the Movidius Stick so the final Graph file could not be generated.
6. Many attempts are made to backpropagate and generate checkpoint files from the **.pb** file but it is beyond the scope of the project and time intensive too.
7. Finally, the project objective of using NCS and detecting Traffic sign was done but not simultaneously due to the above-mentioned constraints.

FURTHER IMPROVEMENTS

1. Generating and compiling the graph file out of the pre-trained Traffic Sign detector network under Tensorflow model and importing it onto the Movidius stick.
2. Due to the unavailability of the checkpoint and meta files, we couldn't generate the graph file. So training our own model can solve this problem with full control over the model files.
3. While running the script to remove distortion, the frame rate of the system is compromised and the overall FPS of the system is reduced.
4. Using new version of Neural Compute Stick i.e Intel Movidius Neural Compute Stick 2 which can deliver up to eight times the performance boost compared to the previous generation Intel Movidius Neural Compute Stick (NCS).

POTENTIAL PITFALLS

Project Part 1

1. **Issue:** Raspberry Pi getting stuck while running the ***make*** command to compile the modules of NCS.
   Fix: In order to compile and run the pre-trained network under Caffe model on Raspberry Pi, we install a lighter version of the NCS which is – "Intel Movidius Neural Compute SDK Python API v1" [3], in order to prevent Raspberry Pi from getting frozen and stuck.
2. **Issue:** Significant drop (by a factor of ~2) in the FPS when image using undistortion as it is more computationally intensive.
   Fix: A multithreading approach to the problem might result in increased FPS.

Project Part 2

1. **Issue:** Python 2.7 version issue while importing different libraries.
   Fix: Compiling and building the libraries(Tensorflow, Protobuf, etc.) from the source through official websites keeping in mind of the compatibility with Python 2.7 version.
2. **Issue:** Obtaining a pre-trained network capable of detecting and classifying the Traffic Signs with a reasonable number of classes of Traffic Signs under Tensorflow model.
   Fix: Since the majority of pre-trained traffic sign networks are only classifiers, finding an appropriate network that does both, classify and detect the signs is a tedious job.
3. **Issue:** Compiling Tensorflow Model networks onto the RaspberryPi.
   Fix: To successfully run a pre-trained network, the minimum requirement is that to look for a model having the *label_map.pbtxt* file and *frozen_inference_graph.pb* file in the GitHub repository.
4. **Issue:** Compiling Tensorflow Model Zoo networks onto a Movidius Stick.
   Fix: In order to compile and generate a graph file and then importing it onto a Movidius stick, we must have *network.meta* and *network.pb* files in the Github repository of the pre-trained network.
5. **Issue:** RaspberryPi throws *Out of Memory* error while loading the present Traffic sign detector model.
   Fix: Increase the swap size of the RaspberryPi from default 1GB to 4GB which can then load the complex Tensorflow model.[13]

REFERENCES

1. https://github.com/markjay4k/movidius-series/blob/master/movidius.ipynb
2. https://github.com/chuanqi305/MobileNet-SSD
3. https://movidius.github.io/ncsdk/ncapi/ncapi1/py_api/readme.html
4. https://www.pyimagesearch.com/2018/02/19/real-time-object-detection-on-the-raspberry-pi-with-the-movidius-ncs/
5. https://www.youtube.com/watch?v=6ZKIjtK6-ug&list=PLX-LrBk6h3wRaT_AEkme7Ne81WTgBzMEg&index=5
6. https://movidius.github.io/ncsdk/tf_modelzoo.html
7. https://medium.com/@kennethjiang/calibrate-fisheye-lens-using-opencv-333b05afa0b
8. http://emanual.robotis.com/docs/en/platform/turtlebot3/raspberry_pi_3_setup/
9. https://github.com/phopley/rodney-project/blob/master/Articles/tensorflow/pi_image.md
10. https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi
11. https://github.com/eumicro/ros_tsd/tree/master/catkin_ws/src/ros_tsr_s4b/models/detection/tsd_full
12. https://movidius.github.io/ncsdk/tools/compile.html
13. https://wpitchoune.net/tricks/raspberry_pi3_increase_swap_size.html