

**MASTER OF SCIENCE IN EMBEDDED SYSTEMS DESIGN**

**Embedded Systems Project Report**

**LANE AND SIGN DETECTION FOR AUTONOMOUS DRIVING (Group-II)**

**Under the Guidance of:**

**Prof. Dr.- Ing. Oliver Prenzel**

*Authored by,*

|  |  |
| --- | --- |
| **Name** | **Matriculation Number** |
| Dhanush Lingeswaran | 38019 |
| Juliet Eldo | 38013 |
| Saranyaa Suresh | 38033 |
| Sayli Deshmukh | 38017 |
| Subhaga Kannan | 38012 |
| Vivek Subramonian | 37995 |

**PREFACE**

As a part of the Master of Science in Embedded Systems Design curriculum and in-order to gain practical experience in the field of Embedded Systems, we are required to work and make a project report on “**Lane and Sign Detection for autonomous driving”.** The basic objective behind doing this project is to get theoretical and practical knowledge on the subject.

In this project we have included various concepts, effects and implications regarding Lane and Sign Detection. Doing this project helped us to enhance our knowledge regarding the parameters and methodologies associated with the project. Through this project we came to know about the importance of team work and the role of devotion towards the work.

Table of Contents:

[**Chapter-1 Inroduction**  5](#_Toc63273248)

[1.1. Introduction: 5](#_Toc63273249)

[1.2. Abstract: 5](#_Toc63273250)

[1.3. Aim and Objectives: 6](#_Toc63273251)

[**Chapter-2 Project Management** 7](#_Toc63273252)

[2. Project Management: 7](#_Toc63273253)

[2.1. Team Chart: 7](#_Toc63273254)

[2.2. Team Communications and Meetings: 7](#_Toc63273255)

[2.2.1. Big Blue Buttons: 8](#_Toc63273256)

[2.2.2. Team communication platform – Git Kraken: 8](#_Toc63273257)

[2.3. Project repository – GitHub: 8](#_Toc63273258)

[2.4. Project Management tool – Project Libre: 8](#_Toc63273259)

[**Chapter-3** 9](#_Toc63273260)

[3.1. System Performance Specification: 9](#_Toc63273261)

[3.2. Use Cases/Use stories: 9](#_Toc63273262)

[3.3. Requirement Specification: 11](#_Toc63273263)

[3.3.1. Definition of the system: 11](#_Toc63273264)

[3.4. Constraints and Limitations: 12](#_Toc63273265)

[**Chapter-4** 13](#_Toc63273266)

[4.1. Software Implementation tools: 13](#_Toc63273268)

[4.2. Hardware Implementation tools: 13](#_Toc63273270)

[**Chapter-5 Lane Detection** 15](#_Toc63273271)

[5.1. Introduction into Lane Detection: 15](#_Toc63273273)

[5.1. Lane Detection algorithm: 15](#_Toc63273274)

[5.2. Canny Edge detector: 16](#_Toc63273275)

[5.3. Hough Transform: 17](#_Toc63273276)

[5.3.1. Hough Transform in OpenCV: 17](#_Toc63273278)

[5.4. Extrapolation of Lines: 20](#_Toc63273280)

[5.5. Selection of required lane lines: 21](#_Toc63273281)

[5.6. Calibration: 21](#_Toc63273282)

[**Chapter-6 Sign Detection** 22](#_Toc63273283)

[6.1. Introduction into Sign Detection: 22](#_Toc63273285)

[6.2. Method 1: Detection with OpenCV and Classification with CNN: 22](#_Toc63273286)

[6.2. 1. Pre-processing: 22](#_Toc63273287)

[6.2.2. Traffic Sign Detection: 22](#_Toc63273288)

[6.2.3. Traffic Sign Detection: 24](#_Toc63273289)

[6.2.4. Traffic Sign Classification: 24](#_Toc63273290)

[6.3. Method 2: Sign detection and classification with CNN: 25](#_Toc63273291)

[6.3.1. Rationale: 25](#_Toc63273292)

[6.3.2. The YOLO algorithm: 25](#_Toc63273293)

[6.3.3. Architecture of YOLO v3: 26](#_Toc63273294)

[6.3.4. Input for the algorithm: 26](#_Toc63273296)

[6.3.5. Detection at 3 scales: 26](#_Toc63273298)

[6.3.6. Grid cell: 27](#_Toc63273300)

[6.3.7. Anchor box: 28](#_Toc63273304)

[6.3.8. Objectness score: 29](#_Toc63273306)

[6.3.9. Class confidences: 29](#_Toc63273307)

[**Chapter-7 Testing** 31](#_Toc63273308)

[7.1. The testing phase: 31](#_Toc63273310)

[7.2. Test Plan: 32](#_Toc63273312)

[7.3. The test Case: 32](#_Toc63273324)

[7.3.1. Lane Detection: 32](#_Toc63273326)

List of figures:

# Chapter-1 Introduction

# (Juliet Eldo)

# 1.1. Introduction into the project:

In the modern day world, road accidents have become very common. They not only cause damage to property, but also keep at risk the lives of people travelling. Road safety is an issue of great concern; it has negative impacts on the economy, public health, safety and the general welfare of the people. These road accidents may be due to many reasons such as rash driving, drink and driving, inexperience, jumping signals, ignoring signboards.

To ensure driver safety and minimize the number of accidents on roads, modern vehicles are employed with driver assistance systems. This advanced system can solve the traffic issues and also it increases the comfort level of passengers. In intelligent transportation systems with improved technologies, the vehicles are made more sophisticated with better infrastructure. But the way to move on the roads by means of lane and sign detection aspect is neglected by many automobile companies and the ways to improve these aspects does not change from many years.

 Due to many external conditions that appear for the lane detection and obstacle detection which may lead to accidents. There are conditions such as appearances such as change of light conditions at night vision, shadows caused by buildings and trees, existence of surrounding objects, mismatching of lanes, and lane changes in curved roads. Lane should have to be detected clearly even with the external factors in consideration. Sign detection will provide driving person confidence even in the different lighting and different environmental situations. One of the main technologies involved is computer vision which has become a powerful tool for sensing the environment and has been widely used in many applications by the intelligent transportation systems.

# 1.2. Abstract:

The field of Intelligent Transport Systems is improving rapidly in the world. The aim of such a project is to develop a fully autonomous vehicle. Lane tracking and sign detection are important aspects of autonomous navigation. This project aims at developing a driving assistance system in the context of autonomous vehicles. The proposed system detects and highlights the boundaries of roads in which the vehicle is travelling and it highlights the sign boards on both sides of the road which act as a warning for the driver. This project  focuses on real-time detecting and tracking of structured road boundaries. Road markings can vary greatly between regions and over nearby stretches of road. Roads can be marked by well-defined solid lines or segmented lines. Real-time and robust automatic traffic sign recognition can support and disburden the driver, which significantly increases driving safety and comfort. For lane detection, the lanes are highlighted inside the user defined region of interest which is done with Hough transform. Two approaches for sign detection are explained. First method is based on the color of sign boards. The red, yellow and blue color of the sign boards are extracted and then detected. The second approach is Yolo method which is a deep convolutional neural method for object detection. Multiple sign boards can be detected and classified with this method.

# 1.3. Aim and Objectives:

The aim of this project is to avoid accidents on roads and to implement a driving assistance system which ensures the safety of the passengers. The system shall detect lanes in any roads with slight curves, in different lighting conditions, shadows and environmental factors and shall detect signs mainly mandatory and precautionary signs.

# Chapter 2 Project Management

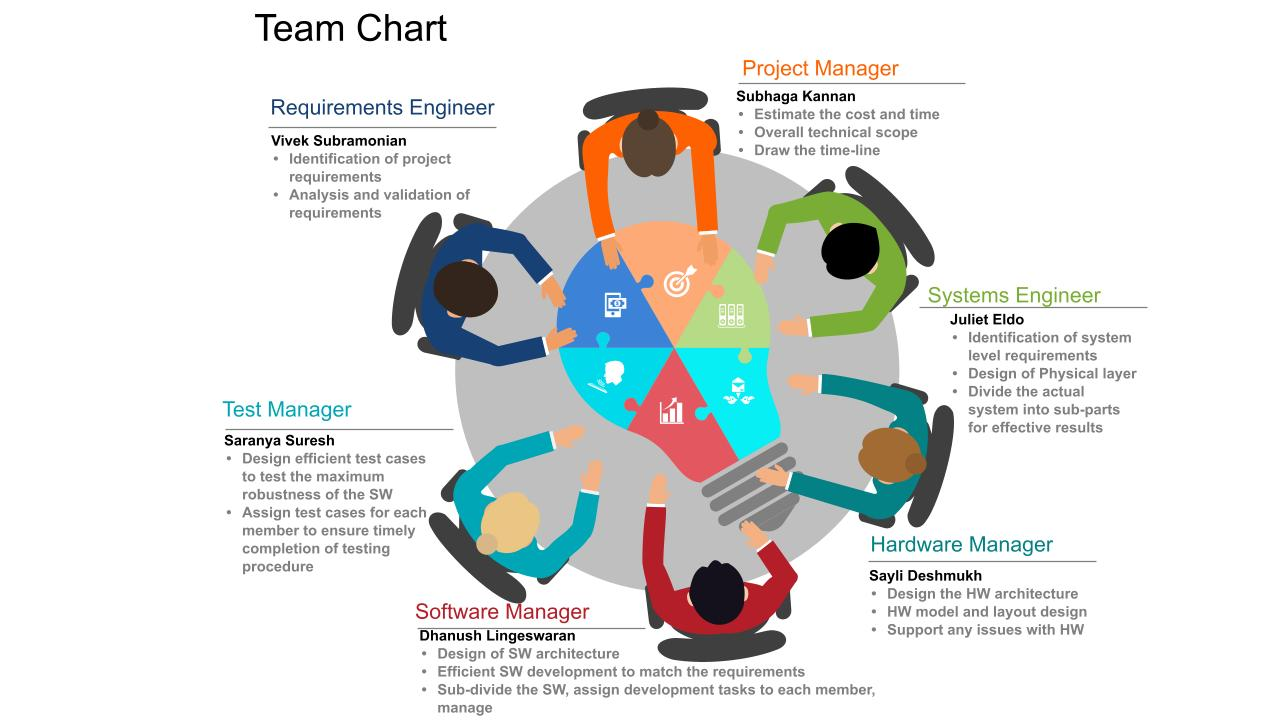
# (Subhaga Kannan)

# 2. Project Management:

The project group consisted of six members from various backgrounds. Each member was assigned a specific role in such a way that each one plays the role of a manager, developer and a tester***.***

# 2.1. Team Chart:

The members were assigned different roles with corresponding responsibility as shown in Fig 2.1. Apart from the role, each one participated in software and hardware development and testing.



*Fig 2.1: Team Chart*

# 2.2. Team Communications and Meetings:

The team meetings held up twice a week, helped in structuring the complex software development was followed throughout the project. The current progress, the tasks for the upcoming week, and any technical support required by any member, were discussed in these meetings. Thanks to them, the team was able to acquire an overview of the workflow. Initially, the requirements were identified with a rank of importance. The requirements were further divided into backlog items, which were assigned to different members of the group. Eventually, in each meeting the backlog items were identified and were documented.

# 2.2.1. Big Blue Buttons:

The Big Blue Button room was used for communication purposes among the members. Online meetings were held up using this application. Apart from the regular meetings, the tool was helpful in having separate meetings among members working for the same topic to discuss the progress.

# 2.2.2. Team communication platform – Git Kraken:

Git Kraken is a visual organisation tool used in software development to easier manage projects. In this project Git Kraken was used as a scrum board to manage backlogs and also keep track of the project's progress. Git Kraken has **integrated cards** that could be used with checklists, this feature was used in the project to get a better understanding of which steps were needed to accomplish the goal of that particular backlog item. Each card had an assignee with a due date and once the tasks were completed, the cards were moved to the ‘**done**’ board.

Link: <https://app.gitkraken.com/glo/board/X7eOs9S39AARKiha>

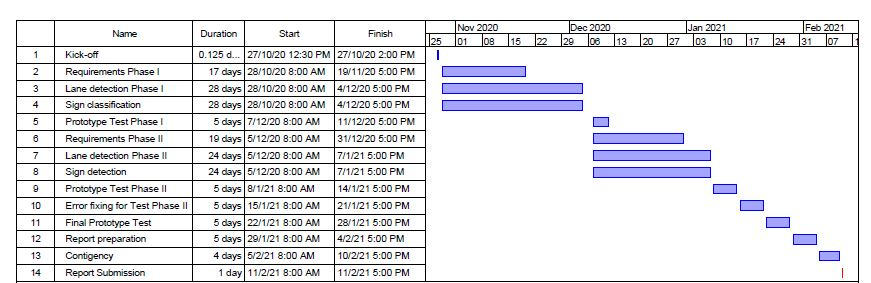
# 2.3. Project repository – GitHub:

A **git repository** was created in GitHub and used for **configuration management** of the project. The project documents, source code files and additional files were uploaded in the repository from time to time. Each update in the project was pushed to the repository, thus maintaining a history of changes made in any file. As an outcome, each member of the team learnt about configuration management and version control.

Link: <https://github.com/DhanushLingeswaran15/ESDproject>

# 2.4. Project Management tool – Project Libre:

Project Libre is a free and open-source project management software system, used to draw the time-line of the project. The time lines were decided using the Gantt chart feature available in the software, shown in Fig 2.2.



*Fig 2.2: Gantt chart with the project time-lines*

# Chapter-3 System Performance Specification

# (Vivek Subramonian)

# 3.1. Overview of SPS:

The System Performance Specification gives a detailed description of how the entire performs and behaves in various scenarios. The main aspect of the SPS documentation are the requirement specifications which enlists in detail, the exact requirements in other words the needs from the customer's side like what a main requirement is, what can be incorporated at the end, what issues needs immediate addressing etc. This in turn describes the performance of the system. SPS documentation begins with the use cases and continues with the Requirement specification.

# 3.2. Use Cases/Use stories:

The use case as mentioned before gives a very brief overview of what the use of a certain main requirement is from the user's point of view. A brief description of use cases related to the project line and sign detection is given below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use case ID:** | UC01 |  | **Use case ID:** | UC03 |
| **Primary Actors:** | The driver |  | **Primary Actors:** | The driver |
| **Secondary Actors:** | The system |  | **Secondary Actors:** | The system |
| **Pre-Conditions:** | System must have a camera attached to it. |  | **Pre-Conditions:** | System must have a camera attached to it. |
| **Main Success scenario:** | Detecting straight lane lines by masking all the noises. |  | **Main Success scenario:** | To detect signs and lane lines while driving in the highway. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use case ID:** | UC02 |  | **Use case ID:** | UC04 |
| **Primary Actors:** | The driver |  | **Primary Actors:** | The driver |
| **Secondary Actors:** | The system |  | **Secondary Actors:** | The system |
| **Pre-Conditions:** | System must have a camera attached to it. |  | **Pre-Conditions:** | System must have a camera attached to it. |
| **Main Success scenario:** | Detecting all major road signs by masking all the noises. |  | **Main Success scenario:** | To detect signs and lane lines while driving on the road of a city. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use case ID:** | UC05 |  | **Use case ID:** | UC07 |
| **Primary Actors:** | The driver |  | **Primary Actors:** | The driver |
| **Secondary Actors:** | The system |  | **Secondary Actors:** | The system |
| **Pre-Conditions:** | System must have a camera attached to it. |  | **Pre-Conditions:** | System must have a camera attached to it. |
| **Main Success scenario:** | To detect signs and lane lines while driving on the road of a rural outskirts. |  | **Main Success scenario:** | To detect signs and lane lines after switching the lanes. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use case ID:** | UC06 |  | **Use case ID:** | UC08 |
| **Primary Actors:** | The driver |  | **Primary Actors:** | The driver |
| **Secondary Actors:** | The system |  | **Secondary Actors:** | The system |
| **Pre-Conditions:** | System must have a camera attached to it. |  | **Pre-Conditions:** | System must have a camera attached to it. |
| **Main Success scenario:** | To detect signs and lane lines while driving during the day time. |  | **Main Success scenario:** | To detect signs and lane lines in all weather conditions. |

# 3.3. Requirement Specification:

The requirement specification is the most important part of the SPS documentation. It is a detailed description of all the requirements derived from the use cases to satisfy all requirements from the customer’s point of view. An Excel file describing the various requirements associated with the project is linked below:



# 3.3.1. Definition of the system:

The system in the requirement sheet refers to a computer with the following hardware specifications:

* Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
* 8GB LPDDR4-3200 SDRAM
* 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
* Gigabit Ethernet
* 2 USB 3.0 ports; 2 USB 2.0 ports.
* Raspberry Pi standard 40 pin GPIO header
* 2 × micro-HDMI ports (up to 4kp60 supported)
* 2-lane MIPI DSI display port
* 2-lane MIPI CSI camera port
* 4-pole stereo audio and composite video port
* H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
* OpenGL ES 3.0 graphics
* Micro-SD card slot for loading operating system and data storage
* 5V DC via USB-C connector
* 5V DC via GPIO header
* Power over Ethernet enabled
* Operating temperature: 0 – 50 degrees Celsius ambient.
* A raspberry pi camera module with 175° wide angle and 5 MegaPixels OV5647 sensor with fish-eye lens.
* A 3.5 inch touchscreen display compatible with raspberry pi with max resolution of 480x320 pixels.

The algorithm will run on python programming language with Tensorflow and OpenCV modules included.

# 3.4. Constraints and Limitations:

Some of the constraints and limitations encountered while formulating the system performance specification are listed below:

* Detection under shadows: When there are shadows i.e, in the presence of inadequate lightning under the influence of shadows it is very difficult to detect both lane lines and signs.
* Detection under different weather conditions: Due to different weather conditions, there might not be sufficient natural light for the pi camera to detect.
* Night time line and sign detection.
* Detection of damaged or faded sign boards: The detection algorithm cannot fully comprehend damaged or faded sign boards since the proper image is not captured. Hence it fails in identifying the sign.
* Detection of traffic lights: Detection of digital signs and traffic signs are not within the scope of the project.
* Curved lines detection: Curved lines over an angle of  20 degrees can be difficult to detect.
* The processor chipset -  Broadcom BCM2711 which has a quad core architecture is sometimes not able to handle sign detection and classification efficiently leading to choppy frame rates in the output display.

# Chapter-4

# (Subhaga Kannan, Sayli Deshmukh)

# 4.1. Software Implementation tools:

# Some of the software tools that the team used for project are as follows:

**1) Pip:** Pip is a package manager for Python packages or modules that the user desires. A package contains all the files, which a developer needs for a module. Modules are Python code libraries, the developer can include in the project.This software developed to receive video streams, detect lanes and signs are based on **Python**. Since Pip is a package management system that includes Python and packages they depend on, this software works as a package manager which makes it easier to work with different packages, Pip has primarily been used to run programs with several different dependencies.

**2) OpenCV:** OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, track camera movements and so on. The lane detection algorithm was developed with OpenCV libraries in this project. This library contains both Canny Edge and Hough’s Transform as functions, which makes OpenCV a simple choice.

[tensorflow, keras, CNN to be updated]

# 4.2. Hardware Implementation tools:

The Hardware specification of the system is as follows :

* Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
* 8GB LPDDR4-3200 SDRAM
* 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
* Gigabit Ethernet
* 2 USB 3.0 ports; 2 USB 2.0 ports.
* Raspberry Pi standard 40 pin GPIO header
* 2 × micro-HDMI ports (up to 4kp60 supported)
* 2-lane MIPI DSI display port
* 2-lane MIPI CSI camera port
* 4-pole stereo audio and composite video port
* H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
* OpenGL ES 3.0 graphics
* Micro-SD card slot for loading operating system and data storage
* 5V DC via USB-C connector
* 5V DC via GPIO header
* Power over Ethernet enabled
* Operating temperature: 0 – 50 degrees Celsius ambient.
* A raspberry pi camera module with 175° wide angle and 5 MegaPixels OV5647 sensor with fish-eye lens.
* A 3.5 inch touchscreen display compatible with raspberry pi with max resolution of 480x320 pixels.

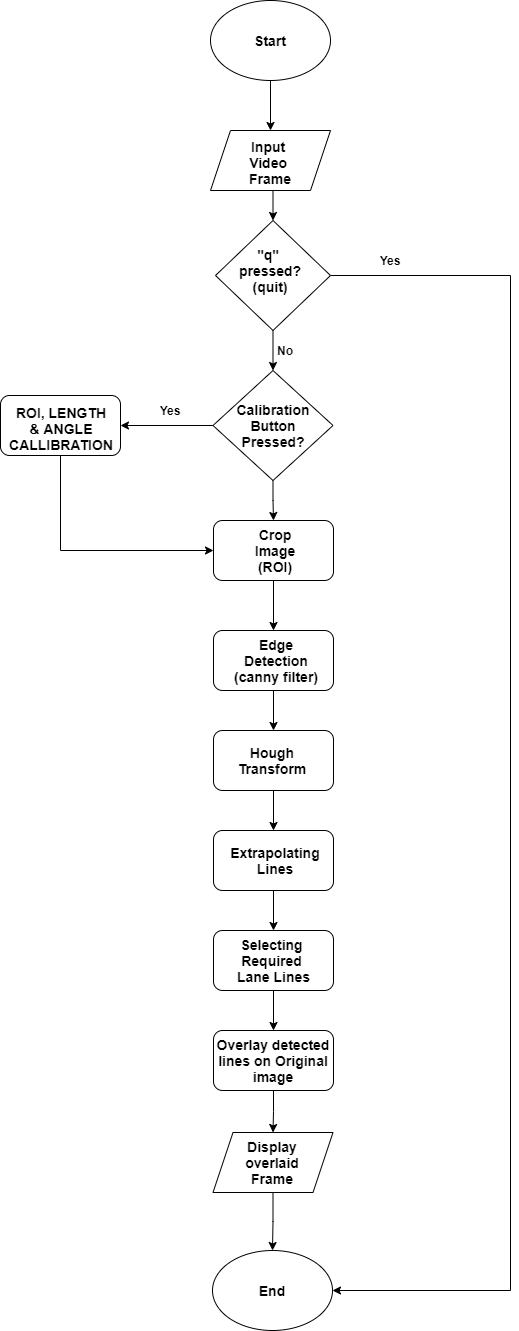
# Chapter-5 Lane Detection

# (Subhaga Kannan, Dhanush Lingeswaran, Juliet Eldo, Saranyaa Suresh)

# 5.1. Introduction into Lane Detection:

An autonomous driving system requires a robust lane detection algorithm in order to prevent the vehicle going off the track. It is critical that the lane detection algorithm is robust against different noises, such as changing weather and light conditions, obstacles and so on. The algorithm captures the video, whose frames are converted into gray scale in each step to reduce the processing time. Then, the edges of the image are extracted using Canny Edge Detector. As the next step, Hough’s Transform is applied to extract the straight lines present in the image.  Eventually, the lane lines are separated using several check mechanisms and displayed. The algorithm detects the new lane, when the vehicle moves from one lane to another.

# 5.1. Lane Detection algorithm:

****

*Figure 5.1: Lane detection algorithm flowchart*

# 5.2. Canny Edge detector:



*Figure 5.2: Canny edge detector on an image*

Edges are significant local changes of intensity in an image, which is useful in finding the boundaries within the image.  Therefore, once the edges of an image are extracted, straight lines can be separated easily from it. Hence Canny Edge Detector was applied to filter all the edges.

In order to detect the edges, many filters like sober filter, canny filter, Gaussian filter and so on are available. Of these, one of the most popular edge detection filters is the canny filter. Canny filter reduces the occurrence of errors, it minimizes the separation of detected edges and real edges and moreover, it has four stages operation which makes it more accurate so that the required edges can be determined properly.

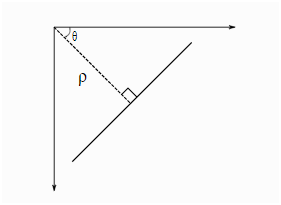
The Canny method uses two thresholds. It disregards all edges with edge strength below the lower threshold, and preserves all edges with edge strength above the higher threshold. The thresholds are specified as a 2-element vector of the form [low high] with low and high values. The four steps of operation of canny filter are:

* Gaussian filter: This in-built filter smoothens the image to remove the noises in the image.
* Calculation of intensity gradients: This can identify the strongest gradient in the image and thus detect that edge for a better result.
* Non- maximum separation: There can be many pixels in an image which might not be considered as edges or which is not thick enough to be an edge. Such pixels are suppressed in this stage.
* Thresholding: Canny filter has double thresholding which makes it different from the other edge detectors.
  + If the intensity gradient exceeds an upper threshold, the filter will accept the pixels.
  + The filter will reject those pixels which are below the lower threshold.
  + There are some pixels which are in-between the upper and lower threshold, these pixels are accepted, provided they are adjacent to a pixel which is above the upper threshold value

# 5.3. Hough Transform:

# The main purpose of Hough transform is to detect the shapes. With Hough transform it is possible to detect all the shapes which can be represented in mathematical form. In this project, the aim is to detect the straight lane lines. Straight lines can be represented in mathematical form as   . Where x and y are the Cartesian coordinate and m and b are the slope and intercept of the line respectively. Similarly in polar coordinate a straight line can be represented as

Where ρ is the perpendicular distance from origin to the line and θ is the angle formed by this perpendicular line and horizontal axis which is measured in anticlockwise direction



*Figure 5.3: Hough transform on a plane*

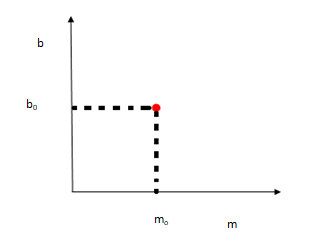
# 5.3.1. Hough Transform in OpenCV:

# Lines in image space can be represented either by equation or a line. This line can be represented as a ‘dot’ or ‘point’ in Hough space or in m and b coordinates. Similarly every line in the image space can be represented in Hough space with a point.

****

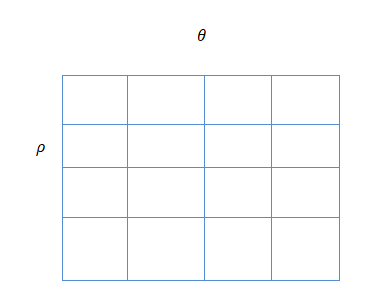
*Figure 5.4:*

In the above figure, the line can be represented by  . In Hough space this line can be represented as follows .

****

*Figure 5.5:*

Hough transform highlights the line with the help of an Accumulator. An accumulator is a 2D array where the x-axis has all possible   values and the y-axis has all possible   values. Any bin in this 2D array corresponds to one line.

****

*Figure 5.6:*

This bin of this array is used to collect evidence about the lines which are there in the image. Each bin has a value and the votes for each bin determine the existence of a line. Opencv provides mainly two functions for detecting Hough transforms which are cv2.HoughLines() and cv2.HoughLinesP(). The syntax for the first one is lines=cv2.HoughLines(image,rho,theta,threshold).

Where,

**Image**          The input image which is 8-bit single-channel binary image.

          Distance resolution of the accumulator in pixels or the accuracy of distance, generally it is taken as 1.

           Angle of resolution of the accumulator in radians or accuracy of angle theta, which is normally taken as /180 which means all possible angles are searched.

**Lines**        : Output vector of lines.

**Threshold**:    Threshold parameter of the accumulator.

The next function for hough transform is HoughLinesP(), which is a Probabilistic Hough transform. The syntax is lines=cv.HoughLinesP(image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]])

The parameters are,

**Image**:Single-channel binary source image.

**ρ** : Distance resolution of the accumulator in pixels.

**Θ** : Angle of resolution of the accumulator in radians.

**Lines**     :   Output vector of lines.

**Threshold**          :    Threshold parameter of the accumulator.

**minLineLength** :    Line segment shorter that minLineLength are rejected

**maxLineGap**     :  The line gaps allowed between the points on the line.

This function is an improvement of the basic Houghlines function. It does not consider all the points rather it requires a subset of random points which is enough for the line detection. It improves the efficiency by providing two more parameters minLineLength and maxLineGap.

# 5.4. Extrapolation of Lines:



****

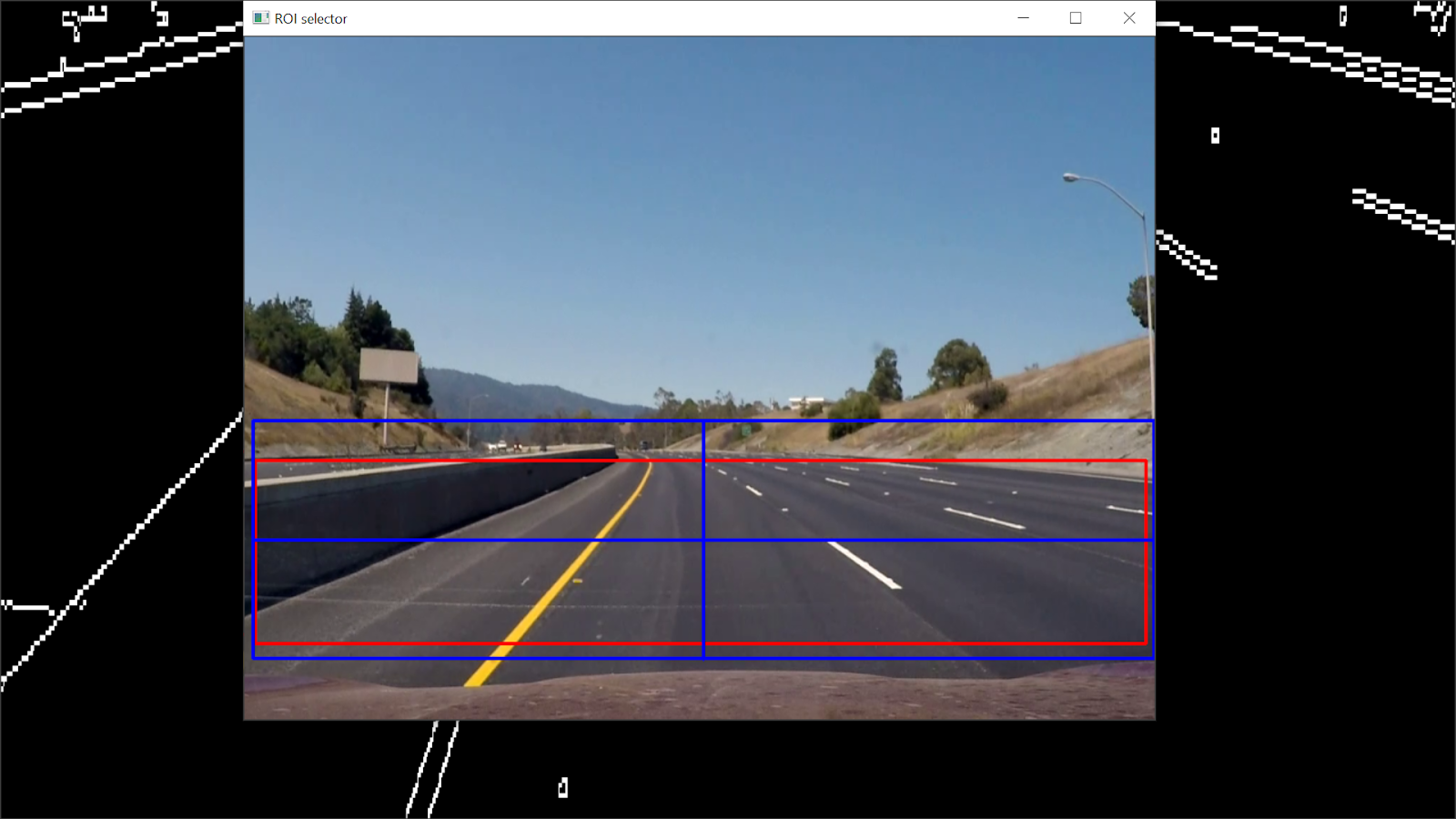
*Figure 5.7 a&b : Extrapolation of lane lines on a road*

# 5.5. Selection of required lane lines:

****

*Figure 5.8: Selection of the required lane lines*

# 5.6. Calibration:

****

*Figure 5.9: Calibration of the ROI*

# Chapter-6 Sign Detection

# (Vivek Subramonian, Subhaga Kannan, Sayli Deshmukh, Dhanush Lingeswaran)

# 6.1. Introduction into Sign Detection:

Two methods have been adapted to detect and classify traffic signs. The first one utilises detection of signs based on color and contour. The system used in this project is a Raspberry pi with a camera module. The major focus is on the detection of traffic signs based on colour detection and segmentation and extraction of the region of interest for classification. Initially, colours are separated and then signs are extracted using mathematical properties of the contours.

# 6.2. Method 1: Detection with OpenCV and Classification with CNN:

# 6.2. 1. Pre-processing:

The videos are real-time captured by the Pi Camera Module, connected to the Raspberry Pi installed in a moving vehicle. Hence, the brightness, contrast, clarity and noises of scenes may have large differences when the weather or other conditions are changed by time and locations. However, these variables could increase recognition difficulty and affect the recognition results. In order to increase the robustness of the proposed scheme, some pre-processes have been used to reduce the influence of variable conditions.

The input frames are read by the camera. The algorithm first converts the image which is obtained from the video feed of the camera from BGR format to HSV, for better colour segmentation, which facilitates the aim of separating red, yellow and blue traffic signs. Then red, yellow and blue channels are separated from the image for contour detection.

# 6.2.2. Traffic Sign Detection:

In general, mandatory and precautionary signs are most important for drivers and their contents contain a red or blue colour frame. Hence, colour extraction is a very effective and efficient solution for selecting candidate sign regions in each frame. However, each colour of the human vision has a fuzzy range in computer colour spaces. The first method extracts the red and blue colour from the frame. After pre-processing, the required colours are extracted using a mask. For this the upper and lower ranges of the mask are defined. The yellow, red and blue range for the mask is given below:

**HSV values for Blue:**

Lower Mask values: [100, 70, 50]

Upper Mask values: [135, 255, 255]

**HSV values for Red:**

Because the red appears in 2 different regions in the HSV colour space, we need 2 separate ranges for it. The same are:

Lower Mask 1 values: [0, 50, 30]

Upper Mask 1 values: [10, 255, 255]

Lower Mask 2 values:[170, 50, 30]

Upper Mask 2 values: [180, 255, 255]



*Figure 6.1: Original image for red sign detection*



*Figure 6.2: Original image for blue sign detection*

Any pixel in the HSV image having pixel values between the above defined range is coloured and all the other regions are coloured black. This image is then separated into 3 channels namely yellow, red and blue channels. Then, all the channels are merged together as a single image.

# 6.2.3. Traffic Sign Detection:

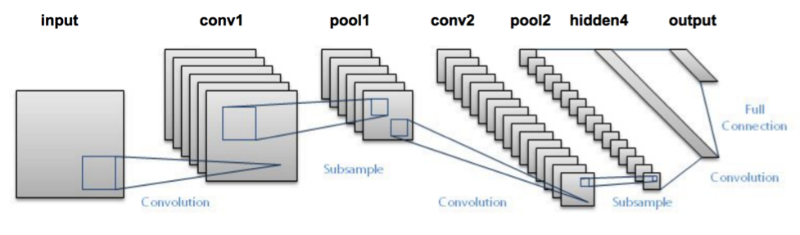
Once the merged image is obtained, the required part which has the sign should be extracted to pass it to the classification algorithm. As the first step, all the contours are detected using an in-built function in OpenCV. Once the contours are extracted, the traffic signs are separated by their area, as they have significantly smaller area than the other contours. This method might also extract some other contours with smaller areas. Hence, a bounding rectangle is drawn around the contours using the function cv2.boundingRect(), which returns x\_position, y\_position, height, width of the contour. Using the ratio of width and height of the contours, all the traffic signs of different shapes like circle, triangle, square and diamond could be fitted inside a square. All the contours with the width and height ratio less than a certain value were discarded. The contours are extracted from the image, once all the conditions are satisfied. The images are then cropped to the size of (60, 60) to pass it into the classifier.

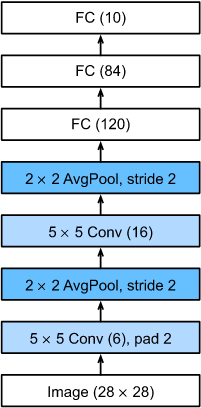
*Figure 6.3 a&b: Extracted signs*

# 6.2.4. Traffic Sign Classification:

Once the traffic sign is localized and cropped using the above mentioned method, the cropped sign image is fed into a simple CNN for classification. The CNN model used here is based on LeNet which has 6 layers.



*Figure 6.4: Convolution layers*



*Figure 6.5: Convolution layer pools*

# 6.3. Method 2: Sign detection and classification with CNN:

# 6.3.1. Rationale:

For object detection and localization using CNN we were in a dilemma whether to use FasterRCNN or YOLO , We tried training a FasterRCNN model but found that the FPS was very low nearly 0.5 , which will get even worse for real time application , so then we chose the YOLO algorithm. It is because unlike FasterRCNN, YOLO is a single shot detector (Darknet as backbone) where it makes classification and bounding box regression at the same time. Whereas in FasterRCNN it has a two-step architecture i.e. it uses RPN and VGG as a backbone model. Well both YOLO and Faster RCNN share some similarities, they both use an anchor box based network structure, both use bounding both regression. In our case we use the YOLOV3 which is extremely fast (45 fps) and accurate for real time detection. Moreover, we could easily trade-off between speed and accuracy simply by changing the size of the model, no retraining required.

# 6.3.2. The YOLO algorithm:

YOLO stands for You Only Look Once. It's an object detector that uses features learned by a deep convolutional neural network to detect an object. YOLO can detect multiple objects on a single image i.e. it predicts the classes and locates the object. It applies a single neural network to the image which divides the image into a number of grid cells and produces probabilities for each cell. After this it predicts the number of bounding boxes that cover some cells region and chooses the best bounding box according to the probability.

# 6.3.3. Architecture of YOLO v3:

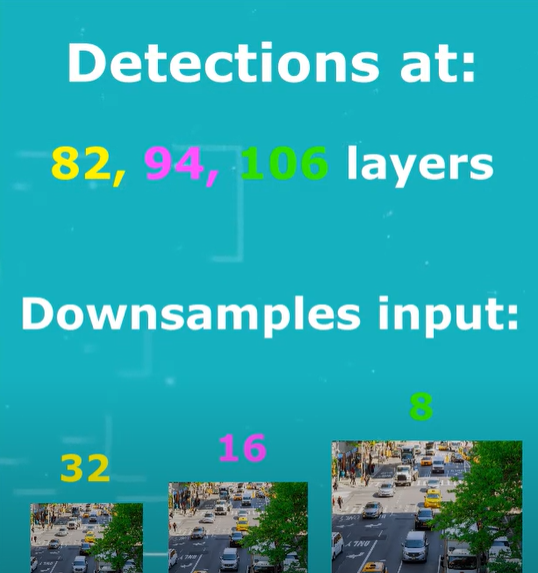
# YOLO v3 uses 53 layers CNN (DARKNET 53) stacked with more 53 layers, producing 106 layers for YOLO v3. The detections are made at 3 layers i.e. at 82, 94, and 106 layers. It incorporates elements like residual blocks, skip connections and up sampling. No form of pooling is used, and a convolutional layer with stride 2 is used to down sample the feature maps. This helps in preventing loss of low-level features often attributed to pooling. CNN layers are followed by batch normalization and Leaky ReLu activation function.

# 6.3.4. Input for the algorithm:

# The input is a batch of images of the shape – (n,416,416,3). Where n is the number of images, 416,416 are the width and the height respectively. The width and height can be changed and should be a number which is divisible by 32. The last attribute is the number of channels (RGB).

# 6.3.5. Detection at 3 scales:

# YOLO v3 makes detection at 3 different scales and places in the network. This happens at layers 82, 94 and 106, the network down samples the input to 32, 16, and 8 at those layers accordingly. The network down samples the input image until the first detection layer, where detection is made using feature maps of a layer with stride 32. Further, layers are up sampled by a factor of 2 and concatenated with feature maps of previous layers having identical feature map sizes. Detection is now made at layer with stride 16. The same up sampling procedure is repeated, and a final detection is made at the layer of stride 8. The following figure explains same:



*Figure 6.6: Detection layers*

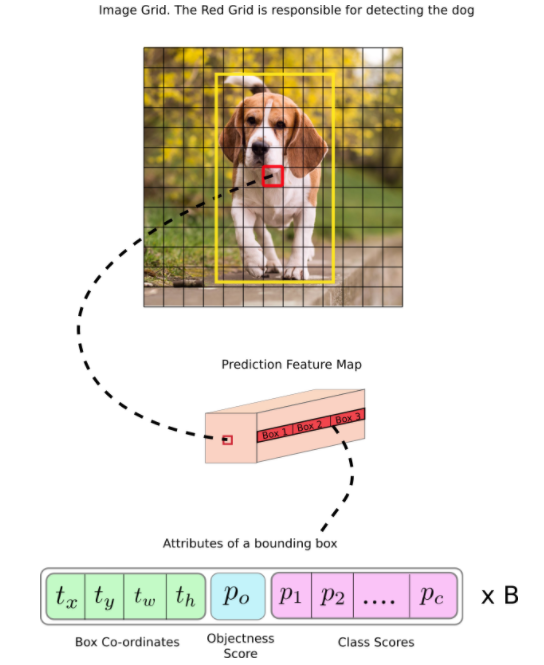
32, 16 and 8 are the network strides. For example, if the stride of the network is 32, then an input image of size 416 x 416 will yield an output of size 13 x 13 (responsible for detecting large objects). Stride 16 will yield an output of size 26 x 26(responsible for detecting medium objects). Stride 8 will yield an output of size 52 x 52 (responsible for detecting small objects). The algorithm applies 1 x 1 kernels at the detection layers to down sample the image. The shape of the kernel has a depth that is calculated by the equation:

                                 (B X (5 + C))

B represents the number of bounding boxes each cell can predict. Each of these B bounding boxes may specialize in detecting a certain traffic sign. Each of the bounding boxes have (5 + C)attributes, which describe the centre coordinates, the dimensions, the objectless score and C(class) confidences for each bounding box. YOLO v3 predicts 3 bounding boxes for every cell. In our dataset we have 42 classes therefore (3 x (5 +42)) results in 141 attributes. Therefore each feature map have the shape of (13 , 13 , 141) , ( 26 , 26 , 141) , (52 , 52 , 141) at the respective detection layer.

# 6.3.6. Grid cell:

# Each cell of the feature map in turn  predicts an object (traffic sign ) through one of its bounding boxes if the centre of the object falls in the receptive field of that cell. (Receptive field is the region of the input image visible to the cell or the kernel). While training YOLO has one ground truth bounding box for one object, when all cells consisting of ground truth are identified the centre cell is assigned to predict the object. The objectness score for this cell is 1. Since all cells including centre one predict 3 bounding boxes each, which one to choose is decided by the anchors.



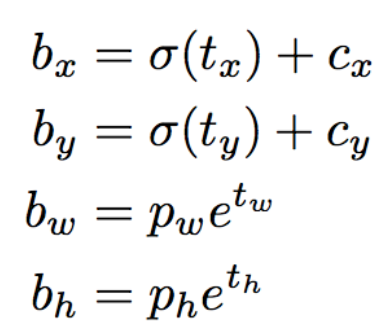
*Figure 6.7: Classification of an object*

# Let us consider an example above, where the input image is 416 x 416, and the stride of the network is 32. As pointed earlier, the dimensions of the feature map will be 13 x 13. We then divide the input image into 13 x 13 cells.

# Then, the cell (on the input image) containing the center of the ground truth box of an object is chosen to be the one responsible for predicting the object. In the image, it is the cell which is marked red, which contains the center of the ground truth box (marked yellow). Now, the red cell is the 7th cell in the 7th row on the grid. We now assign the 7th cell in the 7th row on the feature map (corresponding cell on the feature map) as the one responsible for detecting the dog. This cell can predict three bounding boxes

# 6.3.7. Anchor box:

# To predict bounding boxes YOLO v3 uses predefined bounding boxes that are called anchor boxes (or priors). Anchors help to calculate the predicted BB box real width and height. 9 anchor boxes are used for each scale. i.e. at each scale every grid cell of the feature map can predict 3 anchors. K means clustering is applied to calculate anchors. In practice we predict log space transforms or simply offsets to pre-defined default bounding boxes called anchors. This is done to avoid the unstable gradient problem that arises if we try to predict the width and height of the bounding box. Then, these transforms are applied to the anchor boxes to obtain the prediction. YOLO v3 has three anchors, which result in prediction of three bounding boxes per cell. The bounding box responsible for detecting the object will be the one whose anchor has the highest IoU with the ground truth box. The following formulae describe how the network output is transformed to obtain bounding box predictions:



*Figure 6.8: Formula to obtain bounding box prediction*

Where bx, by, bw, bh are the x,y center coordinates, width and height of our prediction. tx, ty, tw, th is what the network outputs. cx and cy are the top-left coordinates of the grid. pw and ph are anchors dimensions for the box.

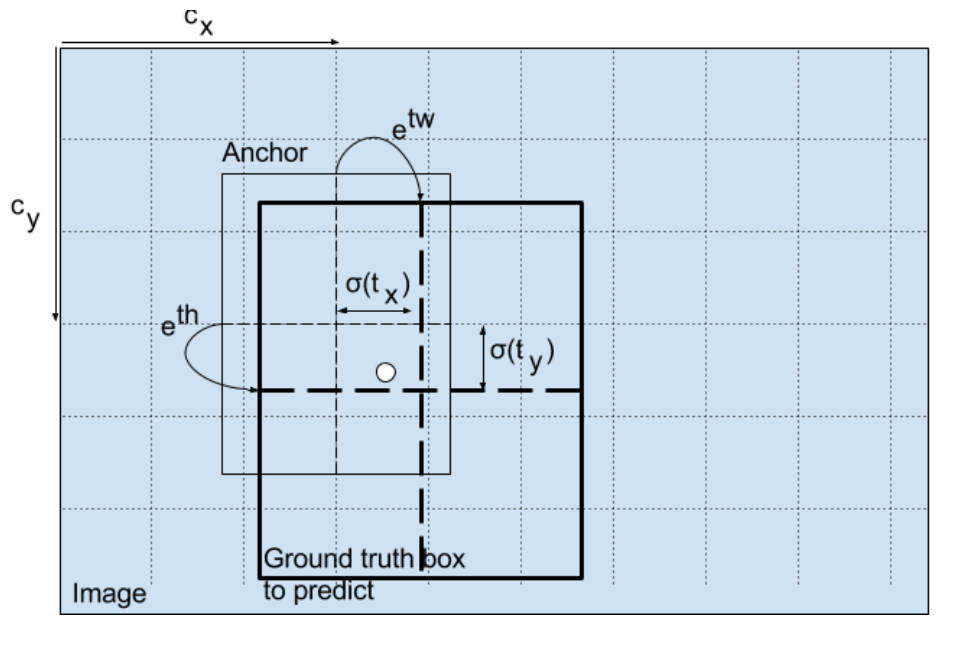
We are running our center coordinates prediction through a sigmoid function; this forces the value of the output to be between 0 and 1. This is done because YOLO doesn't predict the absolute coordinates of the bounding box's center. It predicts offsets which are:

* Relative to the top left corner of the grid cell which is predicting the object.
* Normalised by the dimensions of the cell from the feature map, which is, 1.

For example, if we consider the case of the dog image. If the prediction for center is (0.4,0.7), then this means that the center lies at (6.4, 6.7) on the 13 x 13 feature map. (Since the top-left coordinates of the red cell are (6, 6)). But, what happens if the predicted x,y coordinates are greater than one, say (1.2, 0.7). This means center lies at (7.2, 6.7). Notice the center now lies in a cell just right to our red cell, or the 8th cell in the 7th row. This breaks theory behind YOLO, because if we postulate that the red box is responsible for predicting the dog, the center of the dog must lie in the red cell, and not in the one beside it. Therefore, to remedy this problem, the output is passed through a sigmoid function, which squashes the output in a range from 0 to 1, effectively keeping the center in the grid which is predicting. The resultant predictions, bw and bh, are normalised by the height and width of the image. (Training labels are chosen this way). So, if the predictions bx and by for the box containing the object are (0.3, 0.8), then the actual width and height on 13 x 13 feature map is (13 x 0.3, 13 x 0.8).

# 6.3.8. Objectness score:

Object score represents the probability that an object is contained inside a bounding box. It should be nearly 1 for the centre and the neighbouring grids, whereas almost 0 for, say, the grid at the corners. The objectness score is also passed through a sigmoid, as it is to be interpreted as a probability.



*Figure 6.9: Finding the objectness score*

# 6.3.9. Class confidences:

Class confidences represent the probabilities of the detected object belonging to a particular class. Before v3, YOLO used to Softmax the class scores. However, that design choice has been dropped in v3, and authors have opted for using sigmoid instead. The reason is that Softmaxing class scores assume that the classes are mutually exclusive. In simple words, if an object belongs to one class, then it's guaranteed it cannot belong to another class. However, this assumption may not hold when we have classes like Women and Person. This is the reason that authors have steered clear of using a Softmax activation.

# Chapter-7 Testing

# (Saranyaa Suresh)

# 7.1. The testing phase:

# Testing is the process of evaluating a system and its component with the intent to find whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements. It is important since it discovers defects/bugs before the delivery to the client, which guarantees the quality of the software.

It makes the software more reliable and easy to use. Testing the software ensures reliable and high-performance operation. It verifies that the system meets the functional, performance, design, and implementation requirements identified in the procurement specifications. Thus the testing is conducted for Lane and sign detection.

# 7.2. Test Plan:

# A Test Plan is an important and detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required to perform testing for a software product. It helps to determine the effort needed to validate the quality of the application under test.

# The Test plan contains the description and status about the test, Test environment, Traceability and the important document test cases and its definitions. The test Environment contains the following:

# Evaluation board : A raspberry pi 4

# RAM and Memory: 8GB and 64GB

# Software required: Python with Opencv and Tensorflow modules.

# Required Hardware Tools:

# 1. Raspberry pi  and camera

# 2. Keyboard

# 3. A 3.4 inch Touchscreen monitor.

# 4. A portable power bank.

# The testing is done using different approaches with Video input file and camera input for Lane and sign detection. Thus the test cases are made accordingly. The system is tested in two different scenarios: Autobahn and city Roads. The Autobahn and city roads have different road conditions and signs that are used for certain places.

# 7.3. The test Case:

# The key purpose of a test case is to ensure if different features within an application are working as expected. It helps tester, validate if the software is free of defects and if it is working as per the expectations of the end users. It ensures good test coverage.

# 7.3.1. Lane Detection:

# The testing for Lane detection is carried out to verify whether the system is able to detect lanes under different circumstances. The test is conducted many times to ensure the repeatability of the test. As mentioned earlier, the test is conducted by giving a video input file to Raspberry pi and by using camera input. In the first case, many test videos are collected with different conditions and scenarios.  The videos are fed to the raspberry pi and tested. For the second case, The test video is displayed on the monitor, by focusing the camera on to the monitor it captures the video and processes it. By using these methods a test is conducted to check if the system is able to detect and highlight the lane line. Thus, the different test conducted for Lane detection are: If the system could able to detect Lane lines for the following:

# o   The straight Continuous and Dashed lane lines.

# o   Curved lane lines.

# o   Different Day light Condition.

# o   Shadow Condition.

# o   Night Time period.

# o   Different weather Condition

# o   Different width of the road.

# o   Sudden switching of lanes.

# The test has been conducted for the different scenarios mentioned above and the test is considered to be successful or Pass if it meets up to the criteria of 75% - 85% according to the test case and scenario mentioned briefly in the test case document.