**A**

**Project Report On**

**Advanced driver assistant system**



**SUBMITTED IN PARTIAL FULFILLMENT FOR THE AWARD OF**

**PG DIPLOMA IN SYSTEM SOFTWARE DEVELOPEMENT**

**From C-DAC, ACTS (Bangalore)**

# Guided by:- Mr.Krishna Sir

**Presented by**

**Miss. Tejaswini Patil PRN: 190250130062**

**Mr. Raja Kumar PRN: 190250130073**

**Mr. Parth Sanepara PRN: 190250130084**

**Mr. Ankush Shirbhate PRN: 190250130089**

CENTER FOR DEVELOPMENT OF ADVANCE COMPUTING ACTS-BANGALORE



**CERTIFICATE OF APPROVAL OF PROJECT WORK**

This is to certify that the project report entitled “**ADVANCED DRIVER ASSISTANT SYSTEM**” is a bonafide work carried out by **Miss. Tejaswini Patil ,Mr. Raja Kumar ,Mr. Parth Sanepara ,Mr. Ankush Shirbhate**

in fulfillment for the award of “**PG Diploma in Embedded Systems Design”**

Place: CDAC-ACTS, Bangalore, Feb-2019 Batch.

Signature Signature

(**Mr.Krishna** ) (**Ms.Savithri M** )

Project Guide Course Coordinator

## Acknowledgement

This project would not have been possible without the immense help and support of the CDAC faculty as well as our coordinator. We are in debt to Mr. Krishna our project mentor to guide us through the project with lucid simplicity and skillfulness.

## Contents

Introduction

System Requirements Hardware/Software characteristics Block Diagram

Raspberry pi 2 Applications

1) Lane detection

2) Sign detection

3) Drowsiness detection

References

Introduction

**Advanced driver-assistance systems (ADAS)** are electronic systems that aid a vehicle driver while driving .When designed with a safe human-machine interface, they are intended to increase car safety and more generally road safety.

Most road accidents occur due to human error. Advanced driver-assistance systems are systems developed to automate, adapt and enhance vehicle systems for safety and better driving.

The automated system which is provided by ADAS to the vehicle is proven to reduce road fatalities, by minimizing the human error. Safety features are designed to avoid collisions and accidents by offering technologies that alert the driver to potential problems, or to avoid collisions by implementing safeguards and taking over control of the vehicle. Adaptive features may automate lighting,

provide adaptive cruise control and collision avoidance, pedestrian crash avoidance mitigation (PCAM), incorporate satnav/traffic warnings, alert driver to other cars or dangers, lane departure warning system, automatic lane centering , show what is in blind spots, or connect to smartphones for navigation instructions.

PROBLEM STATEMENT:

To develop image processing algorithms for autonomous vehicle which basically involves:

1. Lane Detection.
2. Vehicle Detection and Recognition.
3. Character, Text and Road Sign Detection and Recognition.
4. Drowsiness detection

## System Requirements

Hardware Requirements

The hardware requirement of the project are as follows:

Computer running Ubuntu/Raspbian as its operating system Raspberry pi 2 (platform)

Miscellaneous

* Logitech webcam C270
* Lan wire
* Micro SD card

## Software Requirements

The software requirements for the project are as follows: Pycharm IDE

Open cv (version 3.2.0) Library Support Python3 Library support

Num py Library Support

## Hardware/Software characteristics Ubuntu/Raspbian Computer

The development of the project requires a Ubuntu/Raspbian operating

system compatible with Pycharm IDE.

## Raspberry pi 2

The **Raspberry Pi** is a series of small single-board computers.

Specification of the Raspberry pi 2

Model : B

Form factor : Standard

Ethernet : Yes

GPIO : 40 pin

Instruction set : Armv7-A (32bit) Memory SDRAM : 1 GB (shared with GPU) USB port : 4 port

Video input : 15-pin MIPI camera interface (CSI) connector, used with the Raspberry Pi camera

Video output : HDMI (rev 1.3) Raspberry pi 2 : Broadcom BCM2836 Soc

Used on 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor



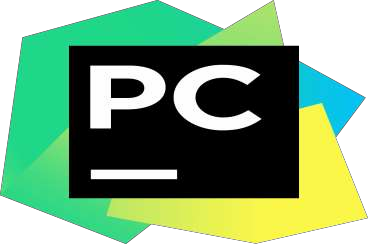
## Miscellaneous

SD card is used to store the OS,python Library, open cv . The SD card slot is present on Raspberry pi 2 board.

## Pycharm IDE

**PyCharm** is an integrated development environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains.[[6]](https://en.wikipedia.org/wiki/PyCharm#cite_note-6)It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django as well as Data Science with Anaconda.[[7]](https://en.wikipedia.org/wiki/PyCharm#cite_note-7)

PyCharm is cross-platform, with Windows, macOS and Linux versions. The Community Edition is released under the Apache License,[[8]](https://en.wikipedia.org/wiki/PyCharm#cite_note-community-8)and there is also Professional Edition with extra features – released under a proprietary license.



## Raspberry pi 2

The **Raspberry Pi** is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote teaching of basic computer science in schools and in developing countries.

The original model became far more popular than anticipated, selling outside its target market for uses such as robotics.

It does not include peripherals (such as keyboards and mice) and cases. However, some accessories have been included in several official and unofficial bundles.

## Steps Used For Detecting Lanes

**Finding Lane Lines on the Road**

Identifying lanes on the road is a common task performed by all human drivers to ensure their vehicles are within lane constraints when driving, so as to make sure traffic is smooth and minimize chances of collisions with other cars due to lane misalignment.

Similarly, it is a critical task for an autonomous vehicle to perform. It turns out that recognizing lane markings on roads is possible using well known computer vision techniques.



## The Pipeline

In this part, we will cover in detail the different steps needed to create our pipeline, which will enable us to identify and classify lane lines. The pipeline itself will look as follows:

* + Convert original image to HSL
  + Isolate yellow and white from HSL image
  + Combine isolated HSL with original image
  + Convert image to grayscale for easier manipulation
  + Apply Gaussian Blur to smoothen edges
  + Apply Canny Edge Detection on smoothed gray image
  + Trace Region Of Interest and discard all other lines identified by our previous step that are outside this region
  + Perform a Hough Transform to find lanes within our region of interest and trace them in red
  + Separate left and right lanes
  + Interpolate line gradients to create two smooth lines

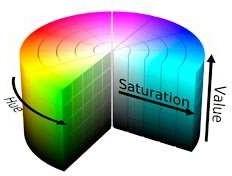
## Convert To Different Color Spaces

While our image is currently in RBG format, we should explore whether visualizing it in different color spaces such as [HSL or HSV](https://en.wikipedia.org/wiki/HSL_and_HSV?source=post_page---------------------------) to see whether they can help us in better isolating the lanes. Note that HSV is often referred to as *HSB* (Hue Saturation and Brightness). I was trying to get my head around the major differences between these two color codes and came across this [resource](http://codeitdown.com/hsl-hsb-hsv-color/?source=post_page---------------------------) the other day which summed it quite well:

HSL is slightly different. Hue takes exactly the same numerical value as in HSB/HSV. However, S, which also stands for Saturation, is defined differently and requires conversion. L stands for Lightness, is not the same as Brightness/Value.

Brightness is perceived as the “amount of light” which can be any color while Lightness is best understood as the amount of white. Saturation is different because in both models is scaled to fit the definition of brightness/lightness.

The diagrams below enable one to visualize the differences between the two:





The image below shows the original image next to its HSV and HSL equivalents



Original vs HSV vs HSL Image

As can be seen while comparing images, HSL is better at contrasting lane lines than HSV. HSV is “blurring” our white lines too much, so it would not be suitable for us to opt for it in this case. At the very least it will be easier for us to isolate yellow and white lanes using HSL. So let’s use it.

## Isolating Yellow And White From HSL Image

We first isolate yellow and white from the original image. After doing so, we can observe how the yellow and the white of the lanes are very well isolated.



Original Image next to HSL Yellow and White Filters

Let’s now combine those two masks using an OR operation and then combine with the original image using an AND operation to only retain the intersecting elements.



Original image next to Combined HSL Filters + Original Image

The results are very satisfying so far. See how the yellow road signs are clearly identified thanks to our HSL yellow mask! Next we move to gray scaling the image.

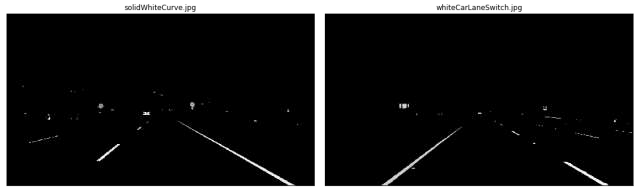
## Convert To Gray scale

We are interested in detecting white or yellow lines on images, which show a particularly high contrast when the image is in grayscale. Remember that the road is

black, so anything that is much brighter on the road will come out with a high contrast in a grayscale image.

The conversion from HSL to grayscale helps in reducing noise even further. This is also a necessary pre-processing step before we can run more powerful algorithms to isolate lines.

powerful algorithms to isolate lines.



Sample Combined HSL Images With Grayscale

## Gaussian Blur

[Gaussian blur](https://en.wikipedia.org/wiki/Gaussian_blur?source=post_page---------------------------) (also referred to as Gaussian smoothing) is a pre-processing technique used to smoothen the edges of an image to reduce noise. We counter- intuitively take this step to reduce the number of lines we detect, as we only want to focus on the most significant lines (the lane ones), not those on every object. We must be careful as to not blur the images too much otherwise it will become hard to make up a line.

The Open CV implementation of Gaussian Blur takes a integer kernel parameter which indicates the intensity of the smoothing. For our task we choose a value of *11*.

The images below show what a typical Gaussian blur does to an image, the original image is on the left while the blurred one is to its right.



Gray scale vs Gaussian Blurred Image

## Canny Edge Detection

Now that we have sufficiently pre-processed the image, we can apply a [Canny Edge](https://en.wikipedia.org/wiki/Canny_edge_detector?source=post_page---------------------------) [Detector](https://en.wikipedia.org/wiki/Canny_edge_detector?source=post_page---------------------------), whose role it is to identify edges in an image and discard all other data.

The resulting image ends up being *wiry*, which enables us to focus on lane detection even more, since we are concerned with lines.

The Open CV implementation requires passing in two parameters in addition to our blurred image, a low and high threshold which determines whether to include a given edge or not. A threshold captures the intensity of change of a given point (you can think of it as a gradient). Any point beyond the high threshold will be included in our resulting image, while points between the threshold values will only be included if they are next to edges beyond our high threshold. Edges that are below our low threshold are discarded. Recommended low:high threshold ratios are 1:3 or 1:2. We use values *50* and *150* respectively for low and high thresholds.

We show the smoothened grayscale and canny images together below:



Grayscale vs Canny Image

## Region Of Interest

Our next step is to determine a region of interest and discard any lines outside of this polygon. One crucial assumption in this task is that the camera remains in the same place across all these image, and lanes are flat, therefore we can identify the critical region we are interested in.

Looking at the above images, we “guess” what that region may be by following the contours of the lanes the car is in and define a polygon which will act as our region of interest below.

We put the canny and segmented images side by side and observe how only the most relevant details have been conserved:



Canny vs Segmented Canny Image

## Separating Left And Right lanes

To be able to trace a full line and connect lane markings on the image, we must be able to distinguish left from right lanes. Fortunately, there is a trivial way to do so. If you carefully look the image (may be easier with the canny segmented images), you can derive the gradient (i.e slope) of any left or right lane line:

* left lane: as x value (i.e. width) increases, y value (i.e. height) decreases: slope must thus be negative
* right lane: as x value (i.e. width) increases, y value (i.e. height) increases: slope must thus be positive

We can therefore define a function that separates lines into a left and right one. We must be careful when the denominator of the gradient (the *dx* in *dy/dx*) is 0, and ignore any line with such line.

In the below images, we colour identified lines belonging to the left lane in red, while those belonging to the right lane are in blue:



Sample Images With Lanes In Different Colors

## Gradient Interpolation and Line Extrapolation

To trace a full line from the bottom of the screen to the highest point of our region of interest, we must be able to interpolate the different points returned by our Hough transform function, and find a line that minimizes the distance across those points. Basically this is a [linear regression](https://en.wikipedia.org/wiki/Regression_analysis?source=post_page---------------------------)problem. We will attempt to find the line on a given lane by minimizing the [least squares](https://en.wikipedia.org/wiki/Least_squares?source=post_page---------------------------) error. We conveniently use

the [*scipy.stats.linregress(x, y)*](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html?source=post_page---------------------------)function to find the slope and intercept of our lane line.

We succeed in doing so, as attested by the following images below:



Images With Lane Lines Identified

Sign detection and object detection

Traffic sign detection and recognition plays an important role in expert systems, such as traffic assistance driving systems and automatic driving systems. It instantly assists drivers or automatic driving systems in detecting and recognizing traffic signs effectively.

Signs and objects that we detected

1. Car
2. Sign ( left,right,stop,pedstrian )
3. Traffic light

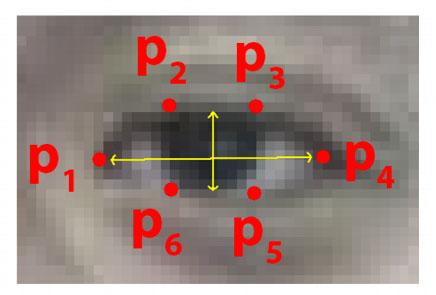
Drowsiness detection

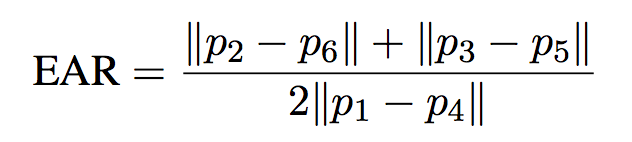
### Description

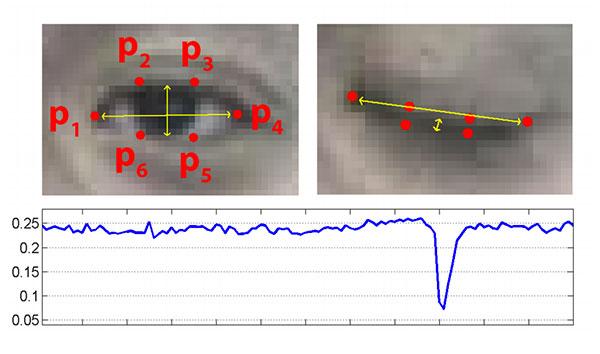
A computer vision system that can automatically detect driver drowsiness in a real-time video stream and then play an alarm if the driver appears to be drowsy.

### Algorithm

Each eye is represented by 6 (x, y)-coordinates, starting at the left-corner of the eye (as if you were looking at the person), and then working clockwise around the eye:.



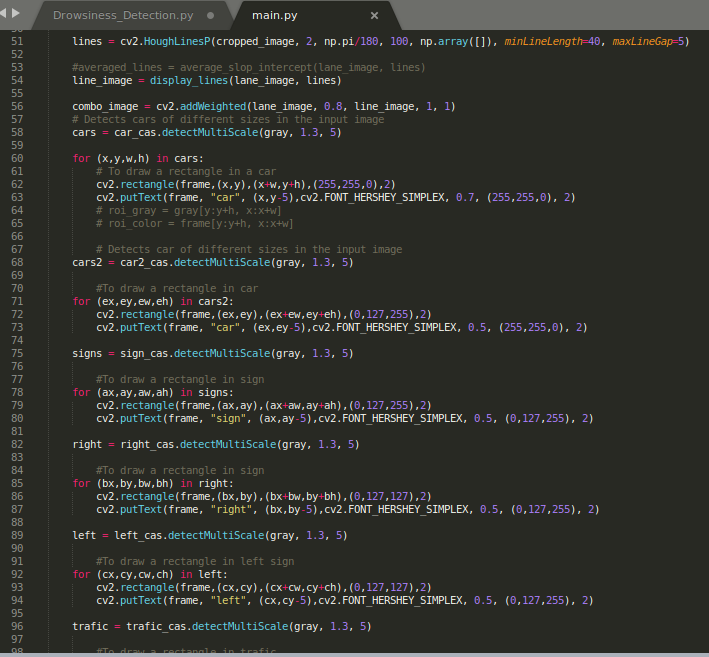


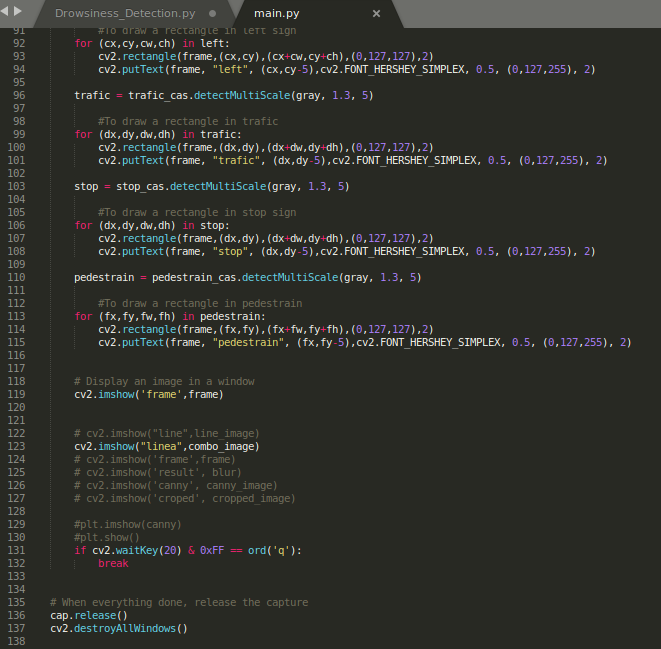


Code snippets

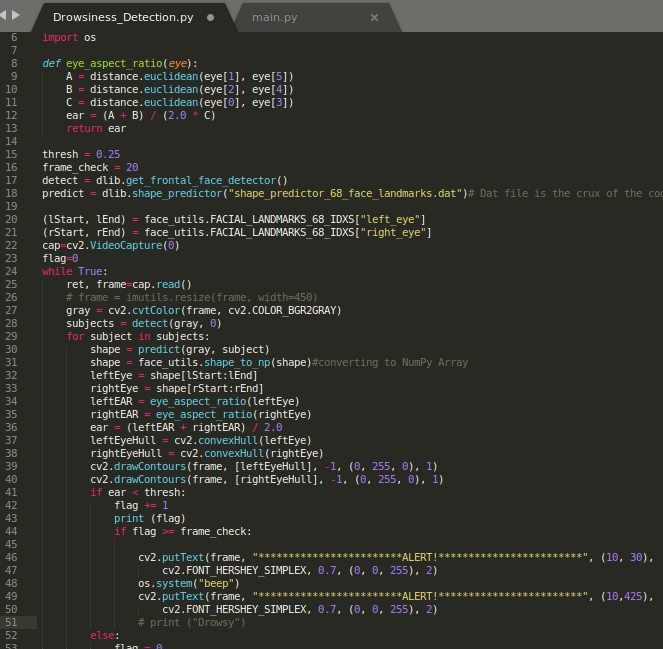
Lane detection and sign detection







Drowiness detection



## Future Improvements

One further step to explore would be to calculate the weighted average of line coefficients in our MemoryLaneDetector, giving a higher weight to more recent coefficients as they belong to more recent frames; I believe frame locality would play a critical role in getting near-perfect lines on video.

We should also consider expressing lines as second degree polynomials or more for examples such as the challenge video.

In the future, I also plan to use deep learning to identify lanes and compare those results against what I obtained with a pure computer vision approach.

Reference

Edureka

Github.com

Open Cv.org

Python.org