“Intellifotainment assist” – A Smart Human-Machine-Interaction (HMI) model for modern Automobiles

31.12.2021

Nishant Wadhwani
Machine Learning Engineer

Overview

Infotainment systems have come a long way since the first set of dashboards installed in cars. In any vehicle, the driver needs to perform several Human-machine-interaction (HMI) actions apart from driving. He/she tends to get distracted from the road while performing secondary tasks such as changing the music track, locking/unlocking the door, adjusting rear view mirrors etc. These secondary actions lead to potential distraction for the drivers, which may cause accidents. In this project, we describe a Human Machine Interaction model for a smart-infotainment system in passenger cars. The model aims to minimize the effort and attention dedicated to secondary tasks, and enable the driver to focus on their primary task, that is, driving. The intelligent infotainment system, or "Intellifotainment-assist" takes into account both critical aspects – convenience and precautionary safety. The purpose of this system is to improve the overall interaction of the driver with the automobile, and also make the system cost-effective. The task is accomplished using robust computer vision and semi-trivial control algorithms.

Infotainment — a fusion of "information" and "entertainment" is the industry-standard term describing the main technological features of a vehicle. This is where all core electronic functions like stereo, navigation, HVAC etc. are controlled and maintained in a certain manner. At the heart of the technology stack, the infotainment system is the main interaction-hub between a human and the vehicle. Evolving from a standard stereo system, the modern infotainment system has numerous features. A few components in a modern infotainment system include -

1. Integrated Head-Unit: is a touch screen based, tablet-like gadget, mounted on the vehicle's dashboard. With user-friendly HMI, the head unit serves as an apt control hotspot for the infotainment framework.

2. Head-up display (HUD): A head-up display is a transparent display that shows data without requiring drivers to look away from the road with the head positioned up and looking forward, instead of angled down. Recent advancement in head-up displays increases safety by allowing users to see everything they need without getting their eyes off the road.

3. High-end DSPs and GPUs to support multiple displays: New age infotainment frameworks are controlled by ground-breaking car processors intended for cutting edge IVI systems. These automotive processors are fit for showing content on numerous showcases (for example Head-up Display or Windshield, Connected smartphones, Head Unit, and many more) and conveys an upgraded in-vehicle experience to drivers and travelers.

4. Smartphone Synchronization: Manufacturers can pair smartphones with automotive systems with the help of Bluetooth connectivity. By doing that, users can access phone features through the infotainment system. For instance, users can oversee approaching, can manage incoming, outgoing, and conference calls, see their contact list, see call logs or even peruse and send emails or SMS.

5. Operating Systems, CAN, LVDS and other network protocol support: Infotainment systems require operating systems that are capable of supporting connectivity, convenience functions, and downloadable software applications to integrate new functions in the system. Operating systems like Android, Linux, QNX and Windows are leading the infotainment segment. The electronic hardware components in infotainment systems are interconnected with certain standardized communication protocols such as CAN (Controller Area Network) which comprises a certain embedded-system OS. CAN or any other network protocol support allows microcontrollers and devices to communicate with each other in applications without the host computer in a reactive way.

6. Multimedia support: Car infotainment systems use Bluetooth, HDMI cable, and USB to transfer audio and video content to display screens, headphones, and speakers. These arrangements can easily display pictures and videos, and they support a broad range of audio formats.

7. Automotive Sensors Integration: Proximity sensors, gesture recognition sensors for detecting ambient light, camera modules and numerous other in-vehicle sensors coordinate with infotainment systems to provide safety-related information to the drivers.

8. Advanced vehicular functionalities: Vehicle infotainment frameworks support features that assists users make the most of their vehicles and improve safety. These features incorporate parking assistance, climate control in the vehicle, daytime running lights (DRL) indicators, and voice assistants that control system functions. For example, a smart parking assistance system can offer users a video feed from the rear view camera, with an efficient optimal algorithm that helps in detecting a particular object which can be displayed on the system screen when reverse gear is being used.

Because of advances in UIs, computational force and show-tech, infotainment frameworks have gotten increasingly wise no matter how you look at it. Autonomous driving has gained momentum over recent times, and the majority of state-of-the-art work focuses in that domain. There are numerous arguments over autonomous vehicles mainly considering safety and ethical laws. We shall not get into a discussion over autonomous vehicles. Our system assumes the requirement of a driver, and is aimed at making the driver's tasks easier.

Along with algorithms, this infotainment system also includes proper hardware system that includes a microcontroller, for proper processing and controlling the devices/equipment according to different inputs feed in the algorithm, servo motors for mirror adjustment scheme and a relay-module for switching between the voltages of variable range.

Goals

Consisting of four distinct modules, our system encompasses –

1. Peripheral control using eye-blinks
2. Compute-efficient attention and drowsiness detection of the driver
3. Automatic rear-view-mirror adjustment system
4. A smart-driving-assist algorithm.

METHODOLOGY:

Hardware set-up and description:

The prototype we have created, incorporating several algorithms and features comprises a dashboard model. The system set-up consists of a camera mounted on the dashboard, in front of the driver, behind the steering wheel. The steering wheel model is attached to the infotainment system via a potentiometer that is attached to the wheel. The wires of the potentiometer are connected to an Arduino UNO which is at the back of the dashboard model. Behind the model, we have a 64-bit laptop, with an Intel i5 processor, running an Ubuntu 18.04 operating system - on which the main video processing is done. When the

model is scaled up for deployment, the laptop will be replaced by a dedicated SOC, enabled with a VPU (Video processing unit) - which is capable of all the required functionalities.

The laptop is connected to the Arduino and webcam via USB cables. On top of the model, is a bulb that represents a headlight. Two other bulbs are attached on either side of the model to represent indicators. This network of bulbs are connected to switches that are fixed in front of the dashboard. A parallel connection is made from the Arduino UNO to the headlight bulb using a 2-channel relay module.

On the top corners of the model, we have attached the rear view mirrors through 2 servo motors mounted on different axes. The mirrors shall have 2 degrees of freedom, and are capable of rotation about two axes (using servo motors). These servo motors are also connected to the Arduino UNO.

Lastly, there is a small LCD screen, which is connected to the laptop through an HDMI cable. This screen represents the HMI screen.

For demonstrating our “Smart-driver-assist” feature, we have performed processing on publicly available test videos. During deployment to an actual automobile, we shall require a camera to be mounted in front of the vehicle, and one behind the vehicle. The front camera shall be enabled when the car is in regular drive mode, and the rear camera shall be enabled when the car is in reverse mode. The live video streams that are supposed to be fed to the system through these cameras, are demonstrated using test videos in the current prototype. The frontal view of our hardware prototype is depicted in Fig-1, with all the distinct visible parts labeled.

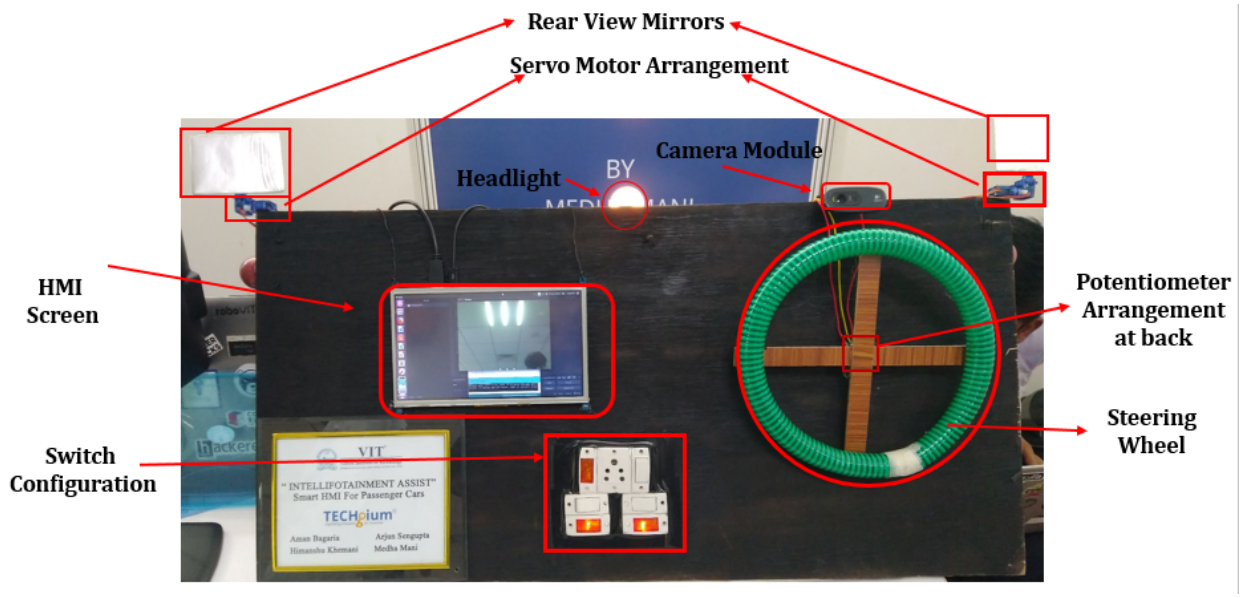


Fig-1: Hardware prototype *

*The processing modules and relay modules are mounted behind the dashboard in the above figure.

In this section, we shall analyze, in detail, each module of the system. The system is broadly divided into four primary modules. The high-level algorithm flow of the entire system stack is depicted in Fig-11. The modules are explained as follows:

Peripheral control using eye blinks:

Specifications

Through a combination of blinks, the driver can turn on or off the headlights, tail lights as well as indicators. To detect single eye blinks, we use python's 'dlib' library, yielding accurate results. Double and triple blinks are identified by measuring the time interval between each blink. In the current prototype, we have demonstrated toggling on and off of the headlight using a double blink. The program disregards involuntary blinks, and takes into account only voluntary blinks that are slightly slower than the involuntary ones.

The method used (adopted by us for single blink detection), is described as follows:

Fig-2 shows the 6 facial landmark points surrounding the eyelids. The method defines a parameter known as eye-to-aspect ratio (EAR) that is calculated using these 6 points, using the following formula:

$$\mathrm{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

Where p1, ..., p6 are 2D facial landmark locations. The numerator computes the distance between vertical eye landmarks.

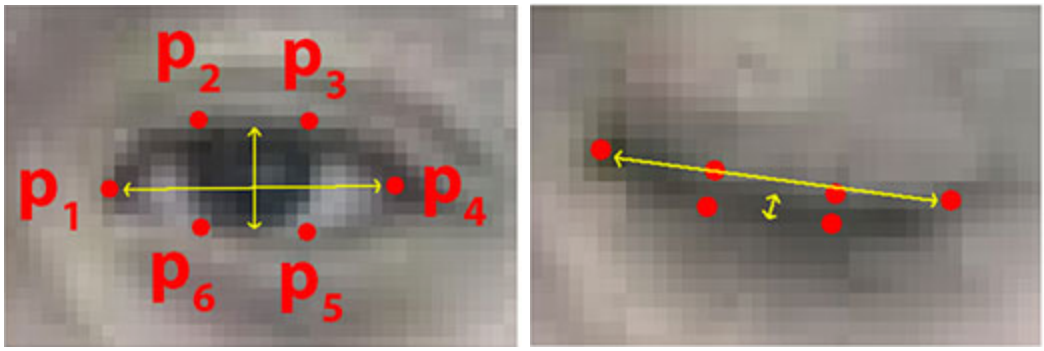


Fig-2: Facial landmarks used to calculate eye-to-aspect ratio (EAR).

Fig-3 shows that the EAR has an almost constant value but decreases sharply while closing the eyelids. This is used to count single blinks.

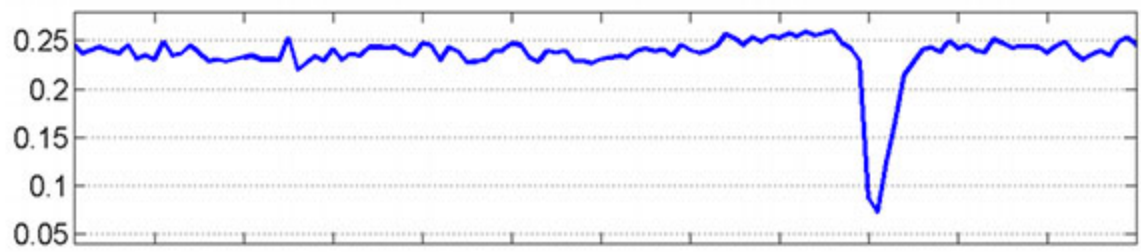


Fig-3: Graph demonstrating the sharp decrease in EAR during blink.

We make a simple modification to this feature by identifying “double blinks” – by setting a time limit between two consecutive blinks. If two consecutive blinks occur within the threshold time, the occurrence of the “double blink counter” is incremented and the headlight is toggled “on” or “off” – depending on the current state. The processing unit sends the toggle information to an Arduino UNO through serial communication. The Arduino controls a relay module through GPIO pins. The relay module is responsible for switching and blocking current to the light bulb. This is how the prototype functions to control the headlight through blinks. The algorithm used in this module is depicted in Fig-4

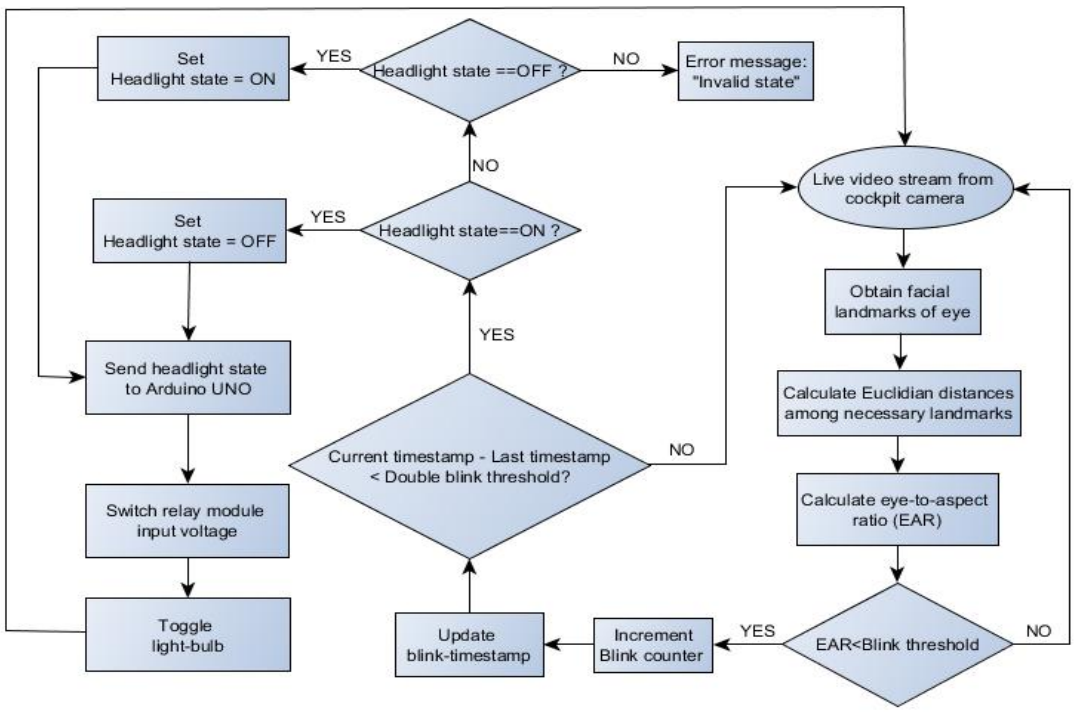


Fig-4: Peripheral control module algorithm flow

Attention and drowsiness detection:

Specifications

The camera is mounted on the dashboard, in front of the driver, behind the steering wheel. Image processing algorithms are applied on this live video stream. The algorithms are described as follows:

a) Iris movement detection: A light weight pre-trained model, known as haar-cascade classifier has been used. They detect a particular class of objects in a frame. We have used

the haar-cascade of an eye to extract the ROI of the eye in frame. This ROI is then converted to grayscale, and binary thresholding is applied. Following a few iterations of erosion and dilation on the thresholded image, the algorithm for direction of sight comes into play. Three coordinates are fixed – to the center, right and left of the frame. Depending on the position of the Iris, pixel values of the left, right and center coordinates vary (as black or white). Correspondingly, the direction of sight is identified as left, right and straight.

b) Facial landmarks: The python library 'dlib' identifies various landmarks of the face – such as eyes, nose, mouth etc. We determine the orientation of the face by measuring the relative distances between these landmarks. Specifically speaking, we calculate the Euclidean distance between points p1 and p4 shown in Fig-2. If this distance is less than a certain threshold value, it means that the driver is not looking straight. This simple concept enables us to qualitatively determine the orientation of the driver's face. As described above, these landmarks are also used to determine the EAR , which quantitatively provides a measure of extent to which the eye is open or closed.

For attention detection, we combine the identified Iris position and orientation of the face (obtained using facial landmarks) to determine where the driver is looking. If the direction of sight is identified to be "off the road" for a period of more than 3 seconds in the moving vehicle, a buzzer alerts the driver to keep his/her eyes on the road.

For drowsiness detection, we simply measure the EAR. If it is found to be less than a certain threshold (which happens just before closing the eyelids) for more than a permissible time limit, the driver is identified to be "sleepy" or "drowsy" and the buzzer rings to give the driver a wake-up call.

Fig-5 depicts the algorithm flow of this module. The points, 'A','B' and 'C', mentioned in the flowchart, are checked for their pixel values. The pixel value 0 corresponds to black color, while pixel value 255 corresponds to white color.

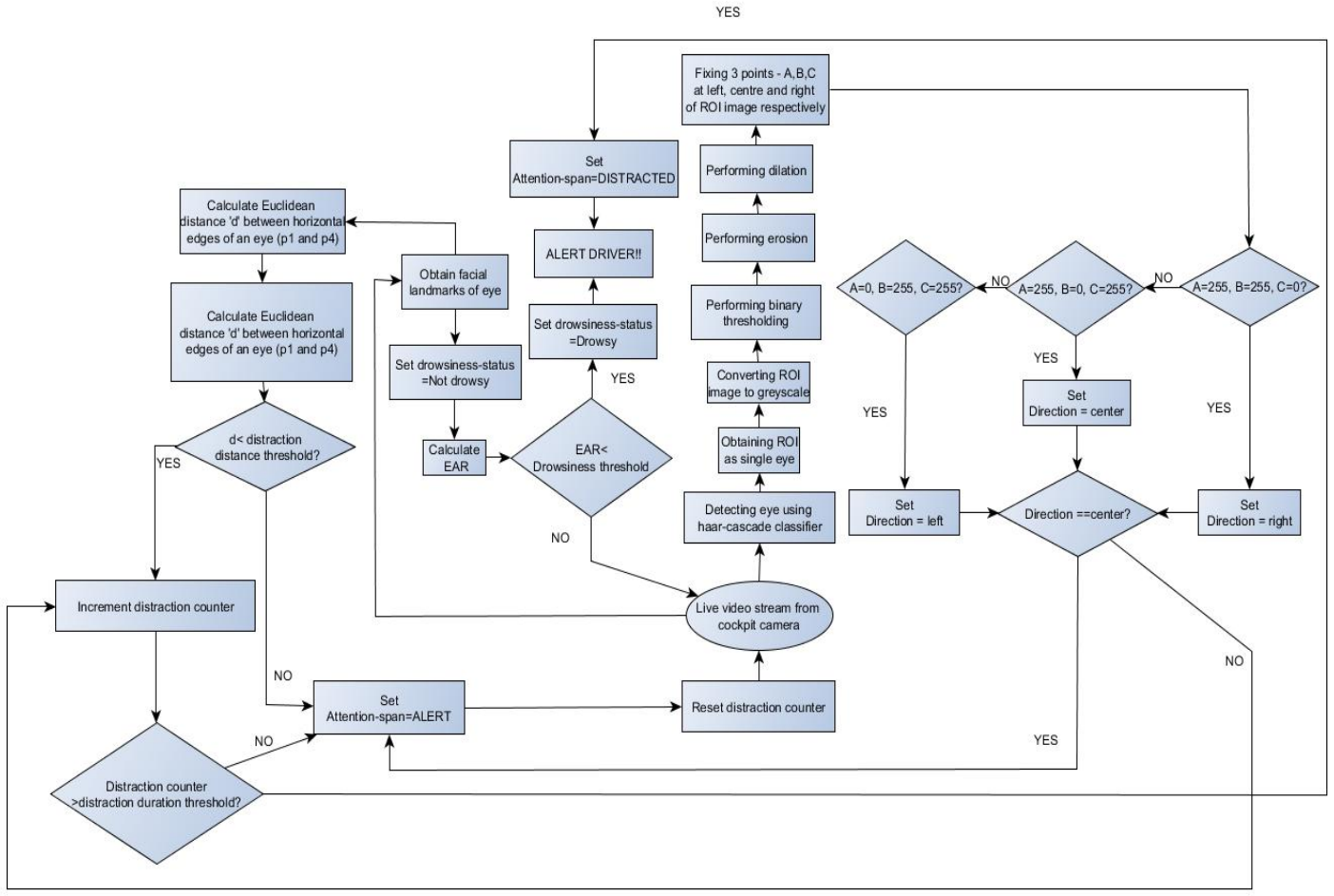


Fig-5: Algorithm flow of attention and drowsiness detection module

Automatic rear view mirror adjustment system:

Specifications

Depending on the 3D coordinates of the driver's face position, the rear-view mirrors are automatically adjusted by the system. The mirrors, mounted on servo motors, adjust their position using the servo motors and do so automatically by identifying the head position. A quantitative mapping is established between the position and servo-movement. A simple PID-controller implemented on the Arduino stimulates the movement of the motors as desired. The algorithm is depicted in Fig-6

Using a haar-cascade classifier to detect a human face, the coordinates of the driver's face position is identified in 3D space. The algorithm used is as follows-

The 2D coordinates are simply obtained by inferring the centroid value of the bounding box in the image's frame. The 'z-coordinate', on the other hand, is obtained by direct correlation with the size of the bounding box. An object closer to the camera, will have a larger bounding box. Using this principle, the depth information of the object is estimated.

Hence we use this simple technique, and obtain 3D coordinates without making use of a depth camera, or monocular depth vision datasets. This significantly reduces overall cost of the system (due to absence of depth camera), and also saves compute (by not using depth datasets) for efficient deployment on embedded devices.

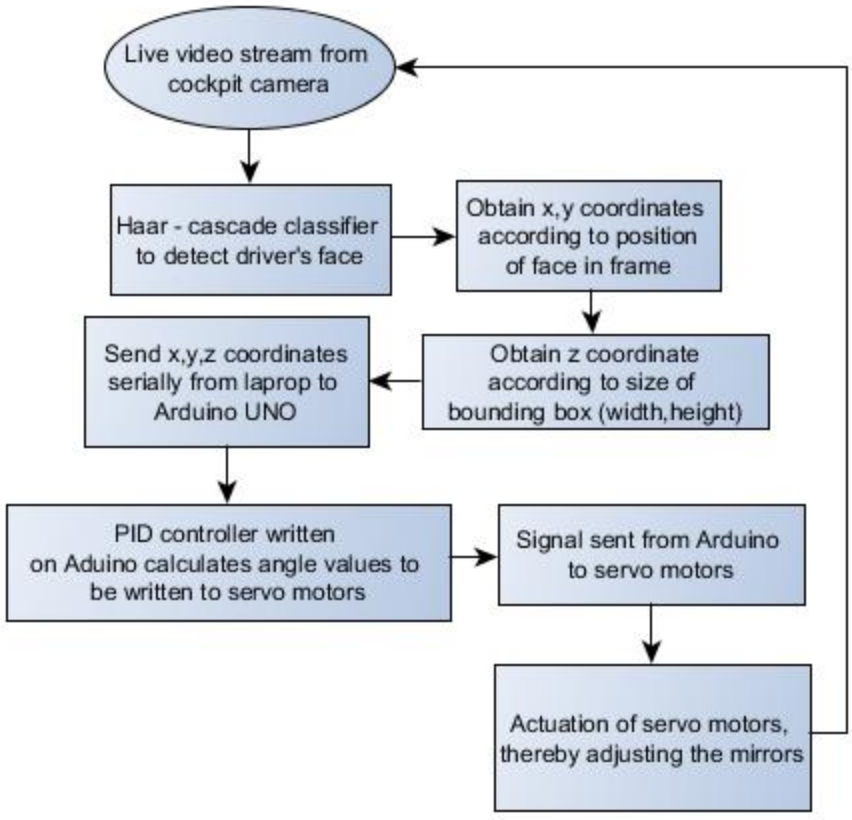


Fig-6: Algorithm flow of automatic rear-view-mirror adjustment

Smart-driving-assist:

Specifications

We propose a Smart-driving-assist feature which uses a combination of lane detection, object detection, and semantic segmentation to find an optimal obstacle-free path, and prompt the driver to steer the car in such a way that it follows that path.

This feature treads along the lines of an autonomous vehicle. It identifies the path to be taken – but rather than performing the actuation itself, it simply prompts the driver to move as instructed. Useful for parking assist, reverse-assist, or regular driving on normal terrain – this feature is extremely beneficial for learners and new drivers. It adds a new dimension of convenience and precautionary safety for experienced drivers as well.

Semantic Segmentation

Semantic Segmentation is a sophisticated computer vision algorithm that associates every pixel in an input image with a class label (i.e., person, road, car, bus or any other object) and categorizes each class based on a specific color component. We have used an E-net (Efficient-neural network) based semantic segmentation architecture which is trained by a huge dataset of different-different environments. It is a low-latency Neural Network that is specifically designed for deployment in embedded and mobile applications. ENet is up to 18× faster, requires 75× less FLOPs, has 79× less parameters, and provides similar or better accuracy to existing models. It has been tested on CamVid, Cityscapes and SUN datasets and yielded optimal inference-time as well as accuracy.

Name	Type	Output size
initial		$16 \times 256 \times 256$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
4× bottleneck1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1		$128 \times 64 \times 64$
bottleneck2.2	dilated 2	$128 \times 64 \times 64$
bottleneck2.3	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.4	dilated 4	$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

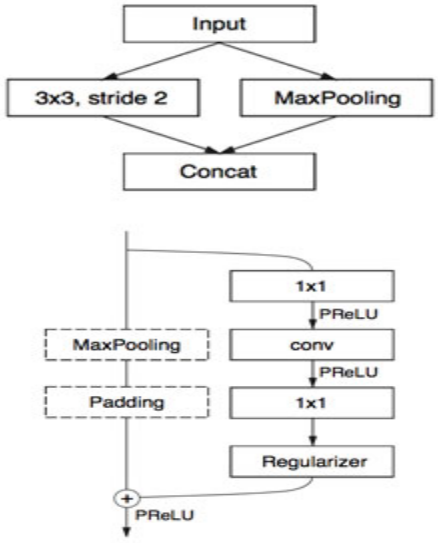


Fig-7: E-net architecture as shown

Object detection

Object detection algorithms start maturing in the last few years, but the competition remains fierce. As shown below, YOLOv4 claims to have state-of-the-art accuracy while maintaining a high processing frame rate. It achieves an accuracy of 43.5% AP (65.7% AP_{50}) for the MS COCO with an approximately 65 FPS inference speed on Tesla V100. In object detection, high accuracy is not the only key component anymore. We want the model to run smoothly in the edge devices. How to process input video in real-time with low-cost hardware becomes important too.

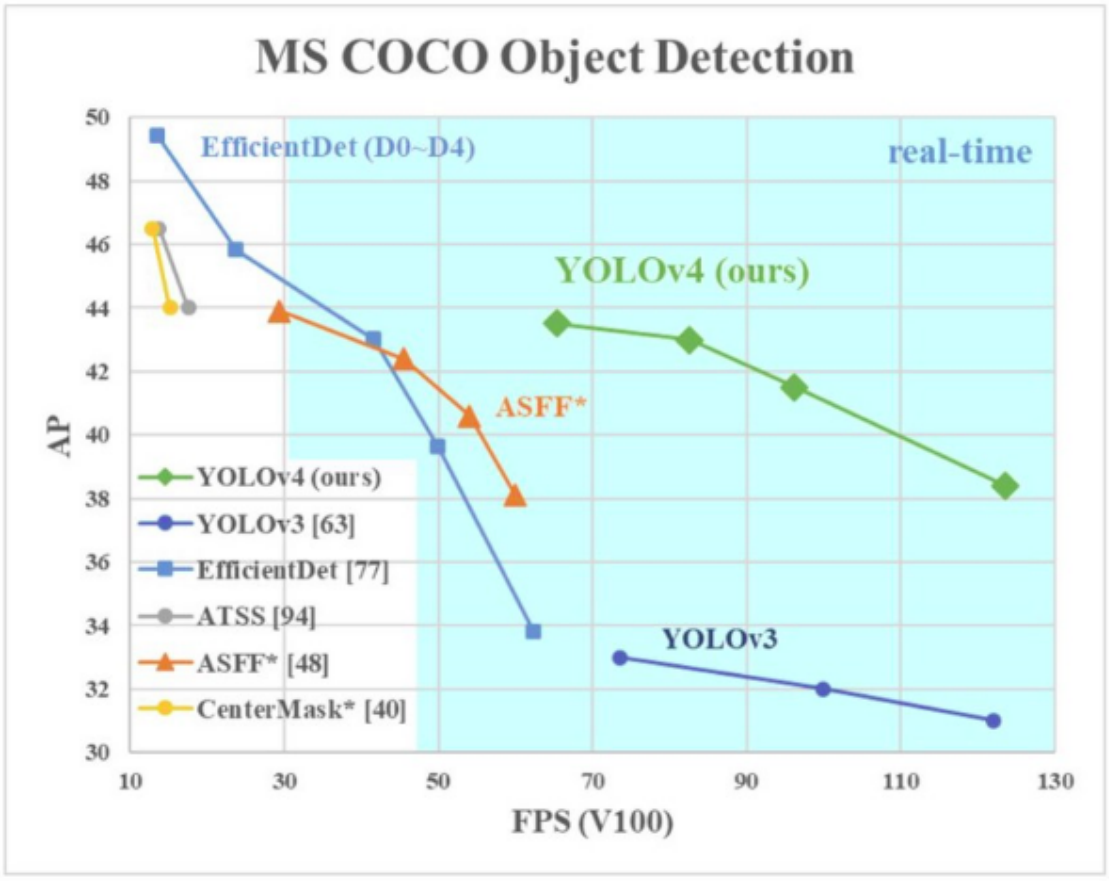


Fig-8: Comparison of the proposed YOLO v4 and other state of the art object detectors

Yolo v4 runs twice faster than EfficientDet with comparable performance. Compared with YOLOv3, the new version of AP (accuracy) and FPS (frame rate per second) are improved by 10% and 12%, respectively.

Bag of freebies (Bof) & Bag of specials (BoS)

Improvements can be made in the training process (like data augmentation, class imbalance, cost function, soft labeling etc...) to advance accuracy. These improvements have no impact on inference speed and are called “Bag of freebies”. Then, there are “Bag of specials” which impact the inference time slightly with a good return in performance. These improvements include the increase of the receptive field, the use of attention, feature integration like skip-connections & FPN, and post-processing like non-maximum suppression.

Backbone

CSPDarknet53

YOLOv4 utilizes the CSP connections above with the Darknet-53 below as the backbone in feature extraction. The CSPDarknet53 model has higher accuracy in object detection compared with ResNet based designs even though they have a better classification performance. But the classification accuracy of CSPDarknet53 can be improved with Mish and other techniques. The final choice for YOLOv4 is therefore CSPDarknet53.

Neck

Object detectors composed of a backbone in feature extraction and a head for object detection. And to detect objects at different scales, a hierarchy structure is produced with the head probing feature maps at different spatial resolutions.
To enrich the information that feeds into the head, neighboring feature maps coming from the bottom-up stream and the top-down stream are added together element-wise or concatenated before feeding into the head. Therefore, the head’s input will contain spatially rich information from the bottom-up stream and the semantic rich information from the top-down stream. This part of the system is called a neck.

Bag of Freebies (BoF) for backbone

The BoF features for YOLOv4 backbone include:

-> CutMix and Mosaic data augmentation,

-> DropBlock regularization, and

-> Class label smoothing

Bag of Specials (BoS) for backbone

1. Mish activation,
2. Cross-stage partial connections (CSP), and
3. Multi-input weighted residual connections (MiWRC)

Bag of Freebies (BoF) for detector

The BoF features for YOLOv4 detector include:

1. CIoU-loss,
2. CmBN,
3. DropBlock regularization,
4. Mosaic data augmentation,
5. Self-Adversarial Training,
6. Eliminate grid sensitivity,
7. Using multiple anchors for a single ground truth,
8. Cosine annealing scheduler,
9. Optimal hyperparameters, and
10. Random training shapes

Bag of Specials (BoS) for detector

The BoS features for YOLOv4 detector include:

1. Mish activation,
2. modified SPP-block,
3. modified SAM-block,
4. modified PAN path-aggregation block &
5. DIoU-NMS

Deep-Sort Tracking Algorithm

Well, we have an object detector that provides us with detections. For tracking, we would require an object tracking algorithm, so that we can have information of the object id which has been avoided by our vehicle.

Deep-sort consists of the almighty Kalman filter tracking it and giving us missing tracks, the Hungarian algorithm associates detections to tracked objects. While SORT achieves an overall good performance in terms of tracking precision and accuracy, also despite the effectiveness of Kalman filter, it returns a relatively high number of identity switches and

has a deficiency in tracking through occlusions and different viewpoints etc. So, to improve this, the authors of DeepSORT introduced a distance metric based on the “appearance” of the object , **The Appearance feature Vector.**

So a classifier is built based on our dataset which is trained meticulously until it achieves a reasonably good accuracy. Then we take this network and strip the final classification layer leaving behind a dense layer that produces a single feature vector, waiting to be classified. This feature vector is known as the appearance descriptor.

Now how this works is that after the appearance descriptor is obtained, use nearest neighbor queries in the visual appearance to establish the measurement-to-track association. Measurement-to-track association or MTA is the process of determining the relation between a measurement and an existing track. So now we use the Mahalanobis distance as opposed to the Euclidean distance for MTA.
DeepSORT is the fastest of the bunch, thanks to its simplicity. It produced 16 FPS on average while still maintaining good accuracy, definitely making it a solid choice for multiple object detection and tracking.

Lane detection involves the following steps:

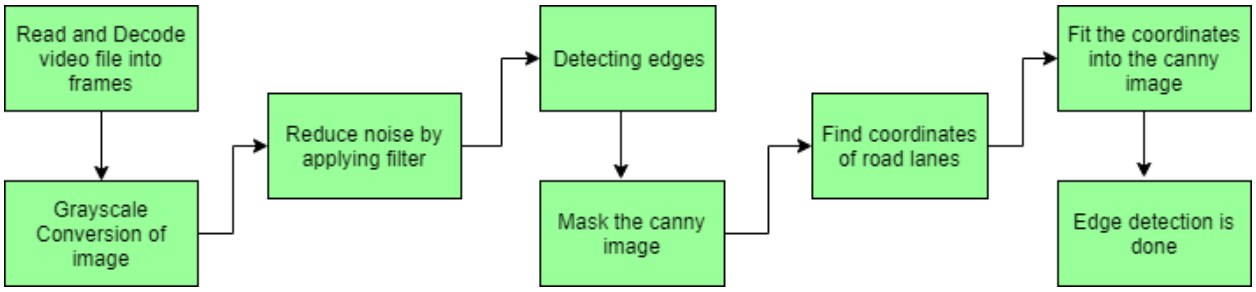


Fig-9: Lane Detection Algorithm

Capturing and decoding video file: We will capture the video using VideoCapture object and after the capturing has been initialized every video frame is decoded (i.e. converting into a sequence of images).

Grayscale conversion of image: The video frames are in RGB format, RGB is converted to grayscale because processing a single channel image is faster than processing a three-channel color image.

Reduce noise: Noise can create false edges, therefore before going further, it's imperative to perform image smoothening. A Gaussian filter is used to perform this process.

Canny Edge Detector: It computes gradients in all directions of our blurred image and traces the edges with large changes in intensity.

Region of Interest: This step is to take into account only the region covered by the road lane. A mask is created here, which is of the same dimension as our road image. Furthermore, bitwise AND operation is performed between each pixel of our canny image and this mask. It ultimately masks the canny image and shows the region of interest traced by the polygonal contour of the mask.

Hough Line Transform: The Hough Line Transform is a transform used to detect straight lines. The Probabilistic Hough Line Transform is used here, which gives output as the extremes of the detected lines

The subsequent steps involved are as follows:

1. Semantic Segmentation to detect roads and obstacles present on the road.

2. For more refinement and better accuracy, we will use yolov4 object detection along with a deep-sort object tracking algorithm. We can take the output of YOLOv4, feed these object detections into Deep SORT (Simple Online and Realtime Tracking with a Deep Association Metric) in order to create a highly accurate object tracker.

3. Lane detection as per road demarcations(if any), finding lane boundary .

4. Warp the detected lane boundaries back onto the original image – and hence obtain the pair of lines that the vehicle should follow

5. Adjust the line pair if any obstacle is detected in the enclosed segment, so as to obtain the obstacle free path.

The lines corresponding to the obstacle free path are thus obtained. Next, we obtain a pair of lines that indicate the trajectory of the car, corresponding to the current position of the steering wheel (as already existing in many modern cars).

Both these pairs of lines are overlaid onto the live video stream from the front or rear camera (depending on whether the car is on regular drive or reverse mode) and displayed on the infotainment/HMI screen. The system prompts the driver to rotate the steering wheel until the two pairs of lines coincide, thereby assisting the driver to follow the obstacle free trajectory and avoiding any possible damage to the car. The algorithm for this module is depicted in Fig-8.

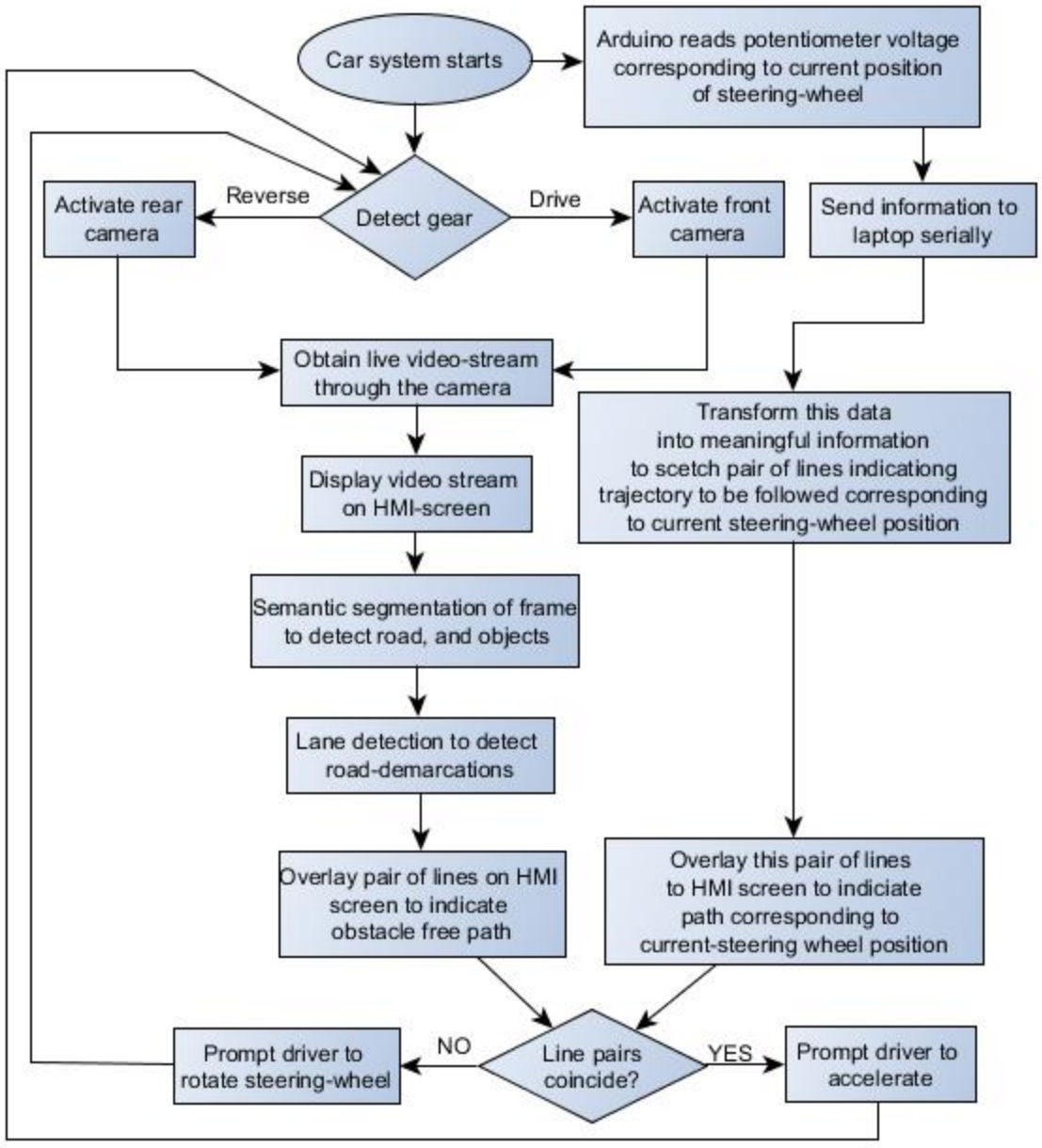


Fig-10: Algorithm flow for Smart-driving-assist module

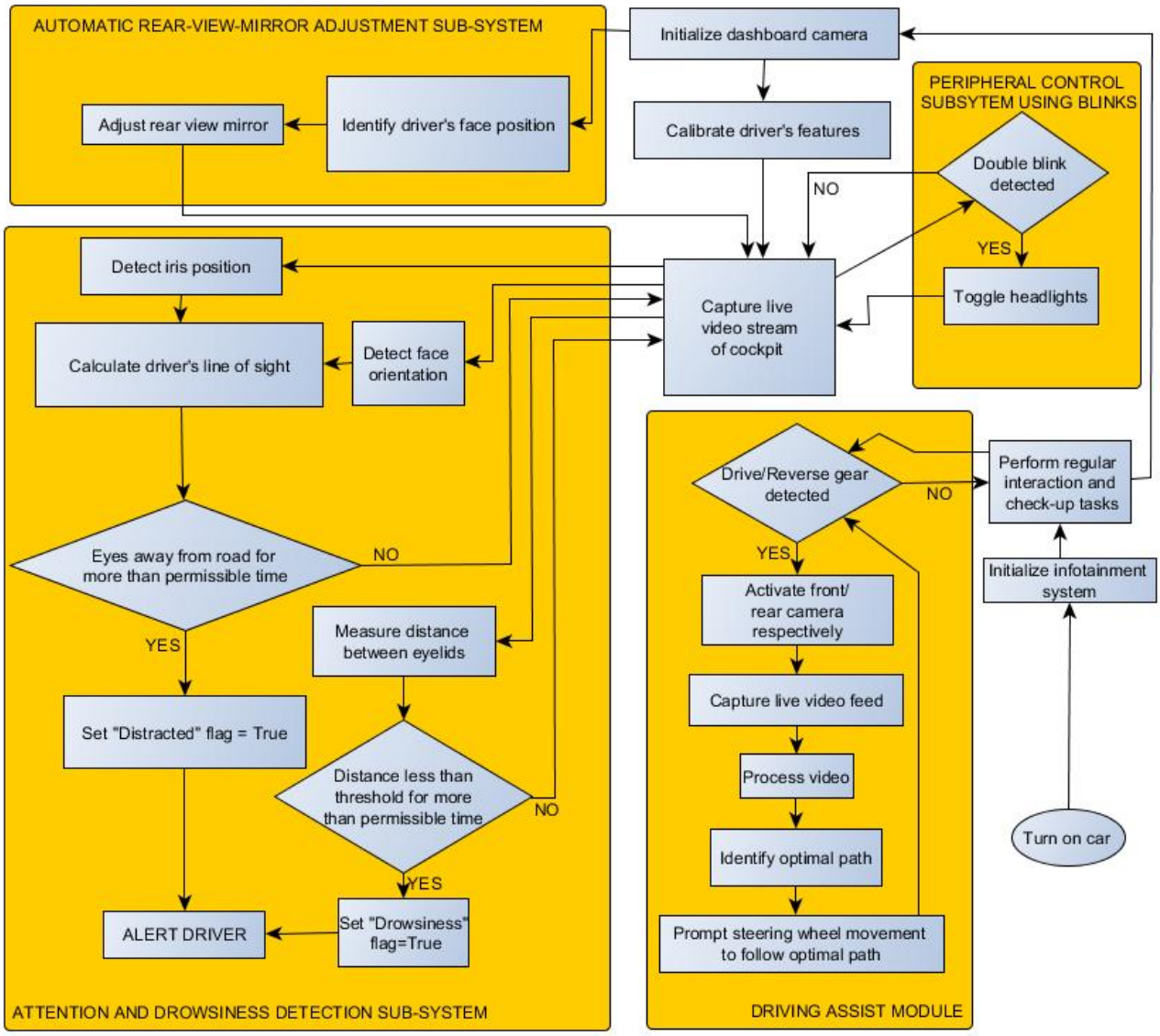


Fig-11: High-level algorithm flow of entire system stack

RESULTS AND DISCUSSIONS:

In this section we discuss the performance of our system, and summarize our results. The live video stream from the vehicle's cockpit, after processing, produces an output as shown in fig-12. This image is one frame of the real-time output that is continuously streamed in the vehicle, and displayed on the infotainment screen. This output is displayed for demonstration purposes. In the deployment set-up, this output may not be displayed on screen – doing which will save a certain amount of compute and hence further reduce overall latency of the system. Even while displaying the output, the current speed of the output stream is over 20 frames per second. Abstaining from displaying the output stream, and performing only the necessary actuations and indications, would further enhance the processing speed.

Most of our discussions in this section shall be based on this output image.

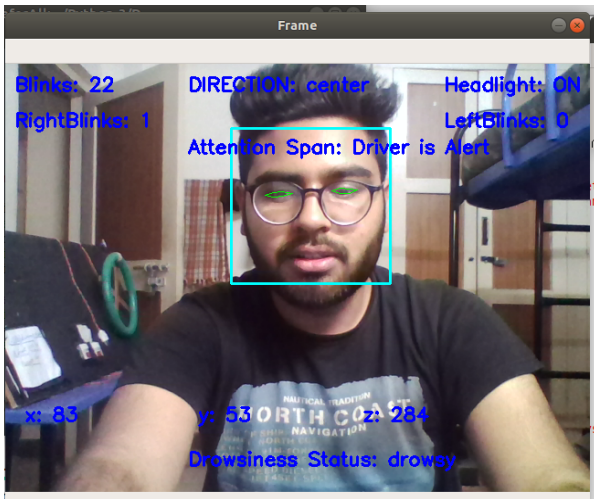


Fig-12: Frame from cockpit camera input, after processing

The top left corner of the frame displays "Blinks" - the total number of voluntary eye blinks since booting of the system. On the top-center, is "DIRECTION" – indicating the direction corresponding to where the driver is looking. The top-right shows "Headlight" – indicating the status of the headlight (ON/OFF). The "RightBlinks" and "LeftBlinks" fields, shown below the fields of "Blinks" and "Headlights" respectively, display the count of "one-eye-blinks" – or "winks" – corresponding to each eye.

Towards the center of the frame, "Attention Span" is displayed – indicating whether the driver is alert or distracted. At the bottom of the frame, we display "Drowsiness Status" – indicating whether the driver is "drowsy" or "not drowsy". Above that, we display the x, y and z coordinates, respectively, of the driver's face-position. Lastly, we draw a bounding box around the driver's face, to indicate the detection, and corresponding to which the coordinates are identified.

All text and drawings are overlaid on the frames using OpenCV functions.

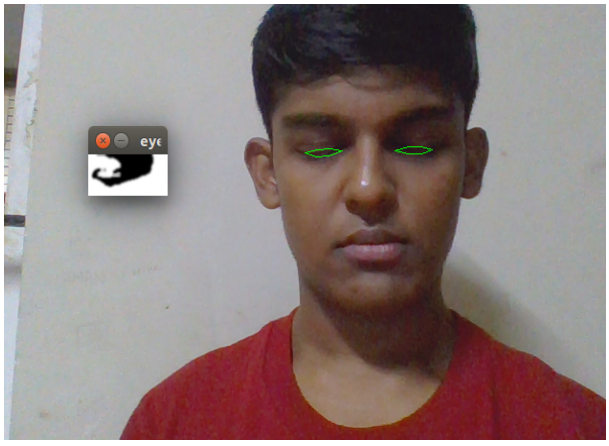
We shall further analyze the results of each module:

Peripheral control using eye blinks:

Milestones

The “Blinks” variable keeps a count on the total number of voluntary blinks. We check the occurrence of a double if two consecutive blinks occur within a time difference of 3 consecutive frames (this threshold may be adjusted depending on the fps value). If a double blink occurs, the headlight is toggled “on” or “off” – depending on the previous state of the headlight.

Fig-13 and fig-14 show the eyelids outlined accurately, for both cases – closed eyelids as well as eyes wide open, using the ‘dlib’ library. Using points on this outline, the EAR is calculated, and the occurrence of blink is detected. The images show that the detection works in well-lit as well as dim-light conditions.



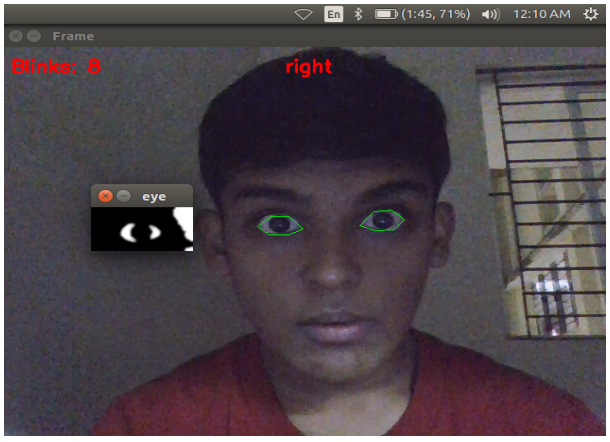
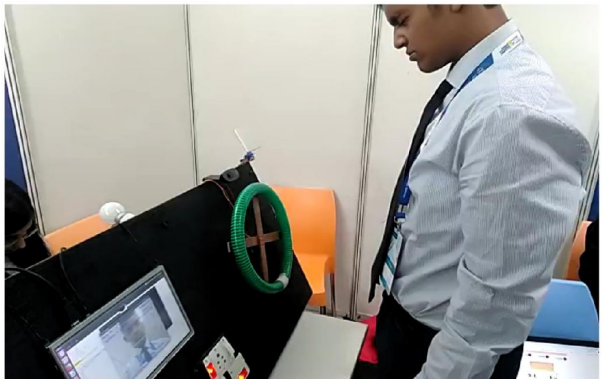


Fig-13: Eyelids closed – identified in a well-lit environment Fig-14: Eyelids open – identified in a dimly-lit environment

Fig-15 and Fig-16 show the results of the output – headlight being toggled on and off (the lightbulb indicating the headlight has been highlighted). The infotainment screen shows the output frames with text and bounding box overlay results as well. This is not clearly visible

in Fig-15 and Fig-16, but the output frame structure is similar to that depicted in Fig-12.



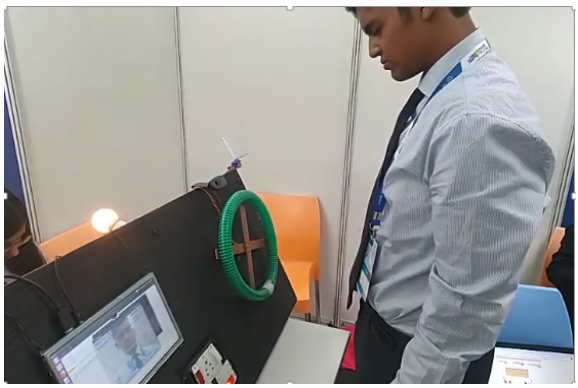


Fig-15: Blinking when headlight is off – to toggle it “on” Fig-16: Blinking when headlight is on – to toggle it “off”

The one-eye-blinks, which we have displayed, are not used for any actuation in the current prototype, but pave the way for future use in suitable systems – such as toggling left and right indicators, wipers etc. The usage of this feature is subject to debate, given that there are many individuals who are physically unable to perform a one-eye-blink.

Overall, we may assert that – once a driver is accustomed to the blink feature, it makes peripheral control virtually effortless.

Attention and drowsiness detection:

Milestones

Using the algorithm mentioned in the methodology, we obtain the results for attention and drowsiness detection.

a) Drowsiness:

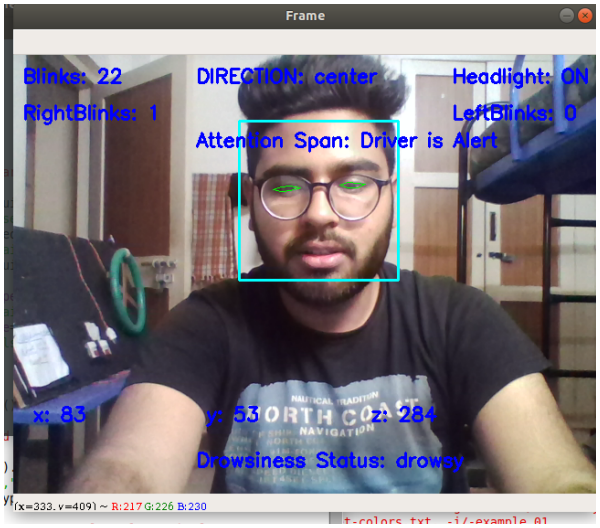
In Fig-12, the subject’s eyelids are closed more than a certain threshold (for a set time-limit of 3 seconds). Hence the system identifies the subject as “drowsy” and displays it. On the other hand, if the subject has his eyes open to a large extent, he has been classified as “Not drowsy”. The threshold distance between eyelids may be tuned. Different individuals may have different thresholds corresponding to their own drowsiness quotient.

To overcome this individuality issue during real-world deployment, the system may ask the driver to make their eyes narrow and wide, before every drive, to tune the drowsiness threshold for the particular driver (Analogous to the way fingerprints are captured on smartphones to set a fingerprint lock).

b) Attention:

In Fig-12, the subject is looking straight and the direction is identified as 'center'. The face is also oriented suitably to completely face the camera. It is therefore inferred that the driver has his eyes on the road, and hence is identified to be 'alert'. On the other hand, if the subject will hold this position for more than the permissible time of 3 seconds, causing the system to infer his state as "distracted" – given that his eyes are off the road for more than the permissible time limit.

Fig-17 and Fig-18 show the drowsiness detection



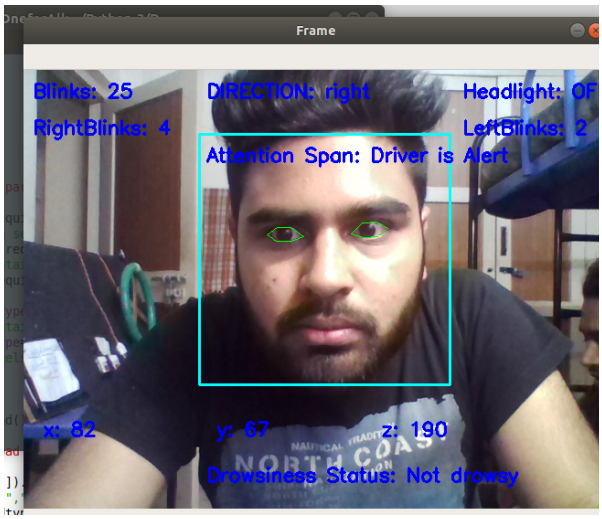


Fig 17: For this distance between eyelids, system is showing drowsy Fig 18: For this distance between eyelids, system is showing not drowsy

Fig-19 and Fig-20 show the iris position detection algorithm, which is the base for our attention detection algorithm, to work accurately even in dim-light conditions.

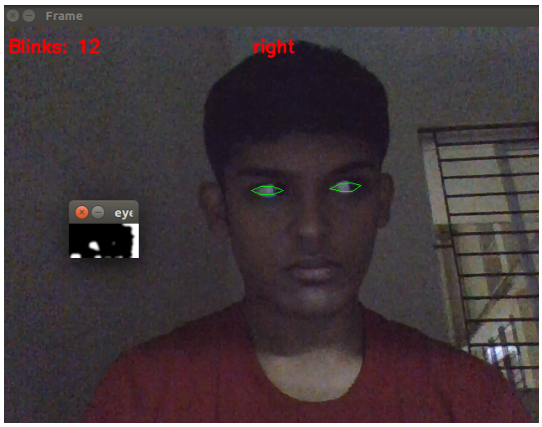


Fig-19: Subject is looking to his right

Fig-20: Subject is looking straight

The current prototype displays the attention and drowsiness status of the driver on-screen. During deployment, loop-closure for the module must be done by adding a buzzer or a speaker that prompts the driver to focus if the system detects him or her to be drowsy or distracted.

Automatic rear view mirror adjustment system:

Milestones

The software returns the 3D coordinates of the driver's face using the algorithm mentioned in our methodology. These coordinates are serially sent to the Arduino Uno. The Arduino Uno then in-turn controls the pair of servo motors on either side of the dashboard, to suitable orient the rear view mirrors (mounted on the servo motors) – using a predetermined mapping. Fig-21 shows the driver's face to be positioned towards the right and left of the frame respectively. The servo motors change the orientation of the mirrors suitably. A slight tilt can be noticed in the orientation of the mirrors between both images. Both mirrors, as well as the position of the driver's face in the frame, are highlighted in the images.

Fig 21: Mirror orientation when driver moves towards his left (right of the frame)

Smart-driving-assist:

Milestones

The semantic segmentation mentioned in our methodology was tried on a test image (Fig-22) and yielded the output as shown in Fig-23. The legend is displayed in Fig-24.

Fig-22: Test image as input

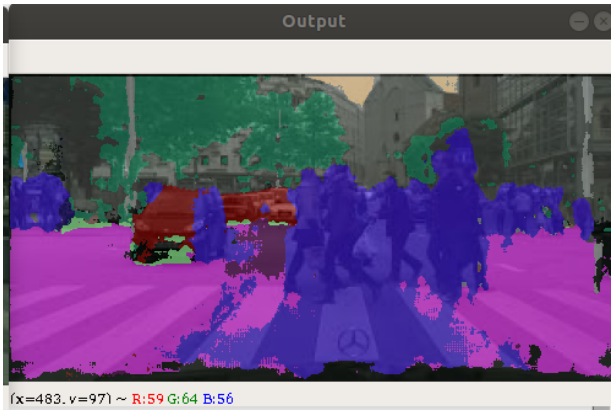


Fig-23: Output after semantic segmentation

Fig-24: Legend for semantic segmentation

The deep-sort object tracker uses YOLOv4 to make the object detections, which the sorting algorithm uses to track as discussed in the methodology part. There exists an official pre-trained YOLOv4 object detector model that is able to detect 80 classes. For our prototype purposes we will use the pre-trained weights for our tracker which is based on CSPDarknet53 architecture and provides good accuracy as well as high frame rate in our embedded system. To implement the object tracking using YOLOv4, first we convert the weights into the corresponding TensorFlow model which will be saved to a checkpoints folder. Then all we need to do is run the object_tracker.py script to run our object tracker with YOLOv4, DeepSort and TensorFlow.

The object-detection and tracking algorithm mentioned in our methodology was tried on a test image (Fig-25) and yielded the output as shown in Fig-26.

Fig-25: Test image as input

Fig-26: Output after detection and tracking

This semantic segmentation together with object detection and tracking, solves the purpose of detecting the road and various obstacles on the road. We combine this with lane detection to find the best obstacle-free path. Lines are drawn suitably to indicate the best obstacle-free path. Fig- 27, 28, 29 and 30 show each stage of the algorithm. The final result, after combination with lane detection, yields the overlay of a pair of lines – as shown in Fig-31 and Fig32.

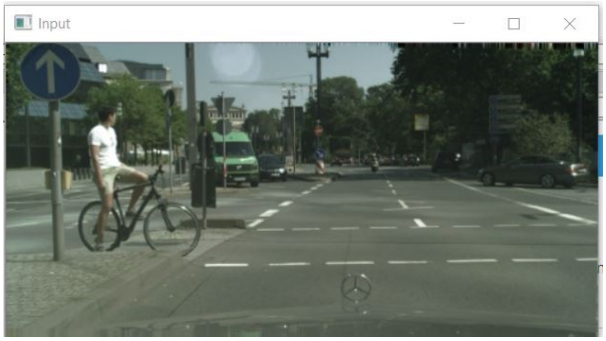


Fig-27: Input image

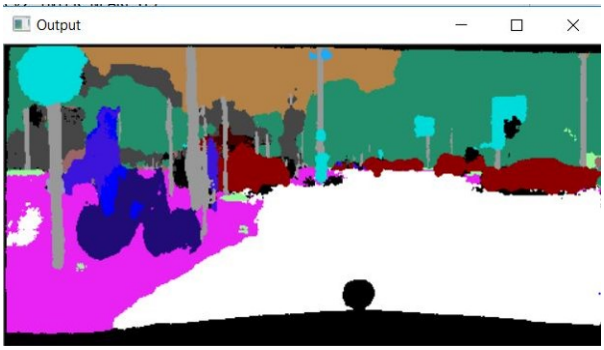


Fig-28: Image after semantic segmentation

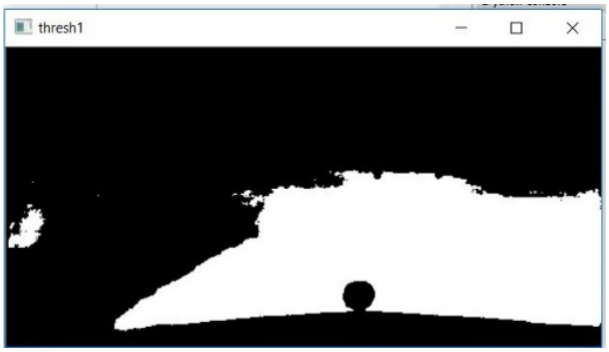


Fig-29: Extracting mask for road

Fig-30: Masked image

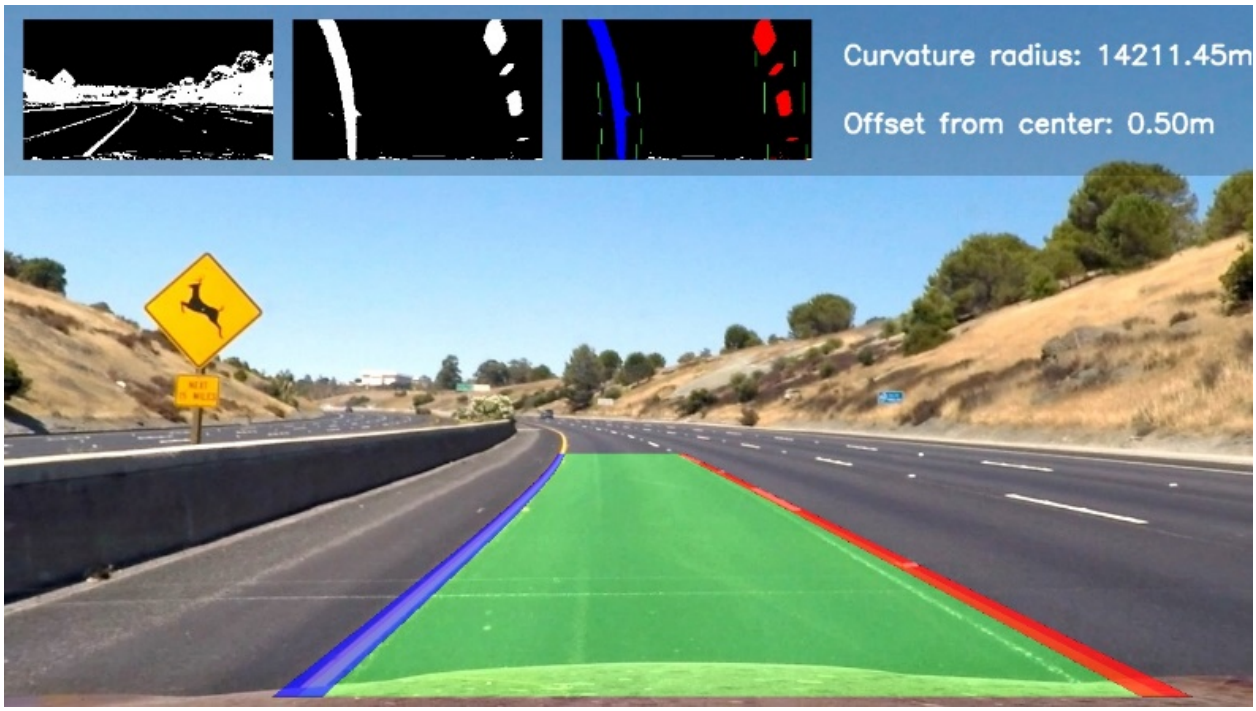


Fig-31: Combining semantic segmentation, object detection ,lane detection, and masking to obtain best obstacle free path for a clear road

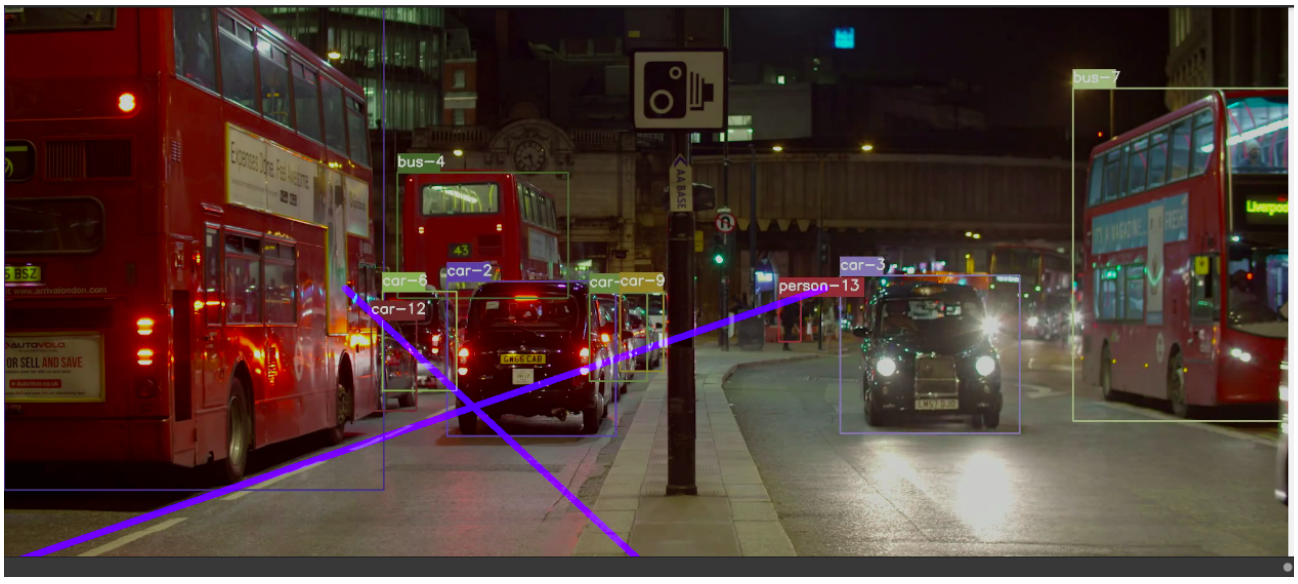


Fig-32: Combining semantic segmentation, object detection ,lane detection, and masking to obtain best obstacle free path for a road filled with traffic

The following images show the results of the lines corresponding to the position of the steering wheel. The steering wheel, connected to a potentiometer, supplies its voltage

readings to the analog input pins of the Arduino Uno in our current prototype. These readings are preprocessed and serial sent to the master device (currently a laptop), which has the openCV interface. Corresponding to the voltage readings, we use openCV functions to draw lines on a black background, thereby indicating the path to be followed by the car, corresponding to the current position of the steering wheel. Fig-33 traces the path corresponding to a steering-wheel with no rotation. Fig-34 traces paths corresponding to steering wheel rotations of various degrees towards the right. Fig-35 traces paths corresponding to steering wheel rotations of various degrees towards the right.

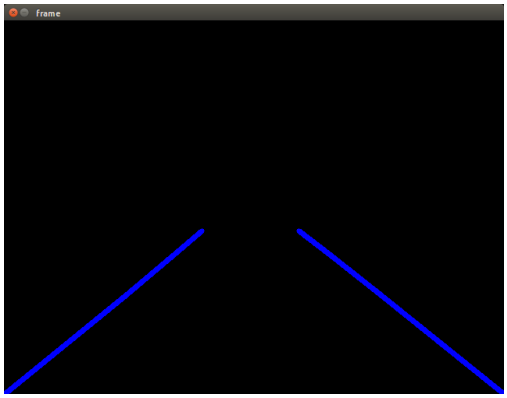


Fig-33: Lines corresponding to straight steering wheel

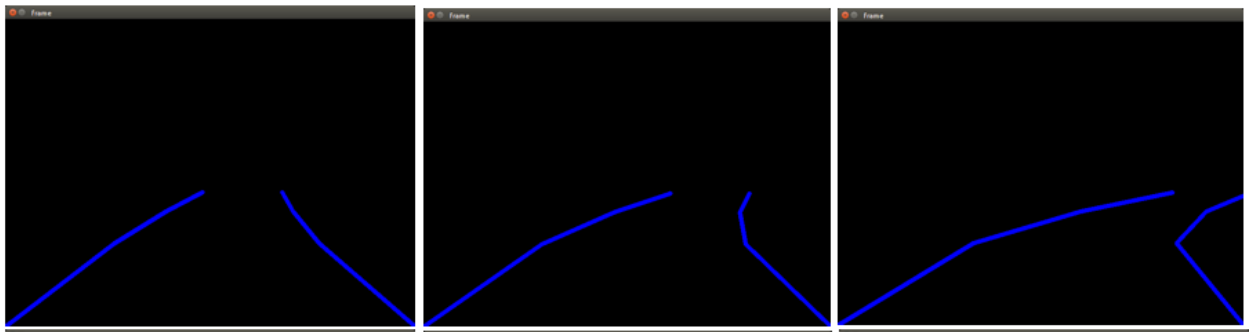


Fig-34: Lines corresponding to steering wheel tilted towards right to various degrees

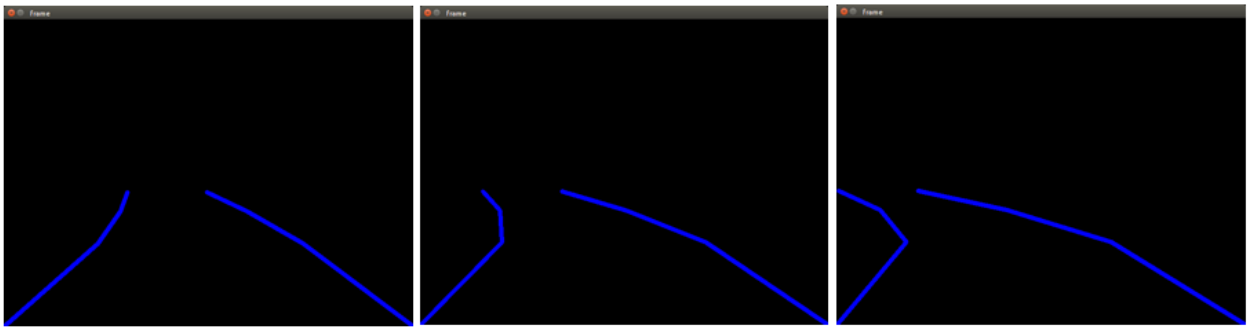


Fig-35: Lines corresponding to steering wheel tilted towards left to various degrees

Our algorithm requires that the path corresponding to the steering wheel position completely coincides with the best obstacle-free path obtained using the semantic segmentation,object detection, tracking plus lane detection algorithm. Hence the above pair of lines are overlaid onto the infotainment screen displaying the best obstacle free path. The system prompts the driver to rotate the steering wheel until the two pairs of lines coincide. The overlay is shown in Fig-36.

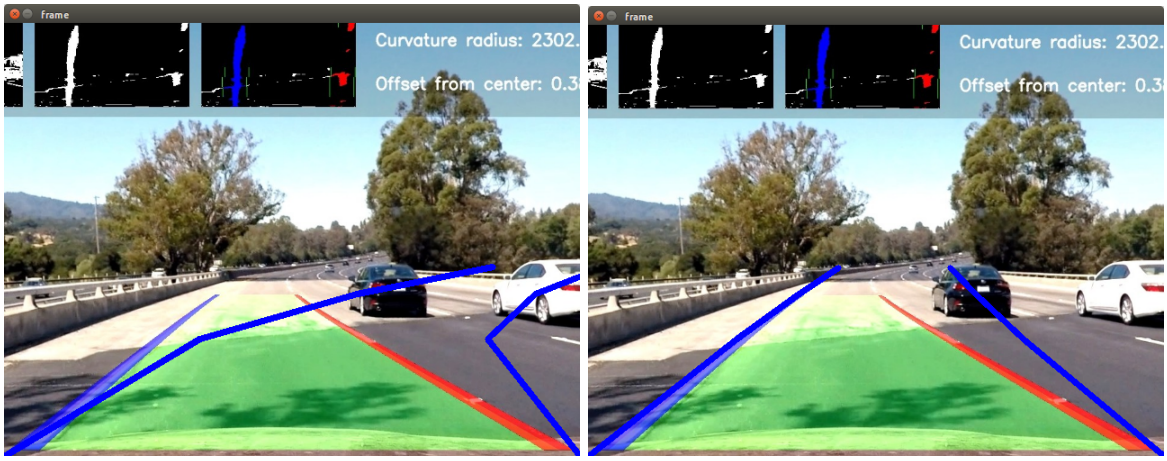


Fig-36: Steering lines overlay on obstacle free path to prompt the driver to turn

The camera extrinsics as well as intrinsics of both - front and rear cameras need to be accounted for during deployment. By weighing in the distortion factors and camera positioning, the line pairs must be sketched.

Not only does this feature add a new dimension of convenience, it also dodges the various laws restricting movement of autonomous vehicles – since our system expects the presence of a driver. Furthermore, we don't make use of any LIDAR, as used by current industry leaders for driving assist, thereby substantially reducing the cost.

Apart from assisting the driver, this feature also gives information about the class of the detected objects and obstacles.

After analyzing the various functionalities of our system, we summarize the pros and cons in Table-1.

PROS	CONS
Use of low-latency algorithms, making our system accurate and quick for real-time results – critical for safety	Different drivers have different facial landmarks, hence the system must be calibrated (to set respective thresholds) by every driver before each drive; the calibration will be automatic, but will account for slight a delay before driving – every time the vehicle starts
No requirement of sophisticated hardware to deploy algorithms; no use of expensive technologies such as lidar – making the system extremely cost effective	Lack of voice command capabilities to control features – which would be even more convenient
Extremely convenient and never before implemented "Automatic rear-view-mirror adjustment system"	Driving assist outputs are not as accurate as results obtained in autonomous vehicles, due to absence of sophisticated technologies such as LIDAR
Completely reprogrammable system; adaptable in different situations	
Minimalistic lines of code due to availability of well-established and open source libraries and platforms such as Open-CV.	
While it treads along the lines of autonomous vehicles due to the driving-assist feature, it does not face any constraint due to safety laws concerned with unmanned vehicles – since our system assumes the mandatory presence of a driver	

Table-1: Comparison chart, enumerating the various pros and cons of the system

CONCLUSION:

While several industry leaders such as Bentley, Aston-Martin, Audi are investing abundant resources for developing smart-driving-assist and semi-autonomous systems, we have taken a holistic approach. We have made an attempt to enhance the overall driving experience-as a whole, by finding room for improvement in modern infotainment systems. Our attempt has not just been focused on giving the driver a better experience, but also to reduce the overall cost of the system-using compute efficient algorithms, to eliminate the requirement of sophisticated hardware. Doing this, will enable large scale deployment of the model into several affordable vehicles, and not just premium supercars.

Such an HMI concept could potentially eradicate the time that the driver devotes to non-driving related tasks and enable him or her to focus on the road. The product shall therefore solve the problem of car accidents and damages to a large extent by automating the secondary features. The HMI shall only alert and assist the driver but never override driving controls from the human.

Our systems add new dimensions to both - precautionary safety measures, as well as convenience. If implemented properly, we are confident that our infotainment-system will reach new heights of HMI and driver assistance technology.

FUTURE SCOPE:

Weighing in the various pros and cons, we can assert that our system - though convenient, cost-effective and deployment friendly, is not perfect. There is definitely scope for improvement. Additional features such as voice-commands to control wipers, car lock, music system and windows may be incorporated - a simple, yet extremely useful idea that would make the life of the driver a whole lot easier. Enabling the driver to speak to his car infotainment system would allow him to control and navigate these functionalities with great ease. The car is enabled with a virtual assistant.

During the initial deployment phase of our system, to prevent errors from creeping in, we will always have a manual override button – for the automatic actuation modules controlling the peripherals. After a good amount of testing, further modifications and refinements can be made.

There’s no denying the fact that infotainment systems pave the future path for the automotive industry. Convenience and safety of the customers are of prime importance in this sector. We are witnessing a dawn of intersection between the automotive and computer industries. Though this is primarily evident in electric cars, and autonomous vehicles, it is also distinctly prominent in infotainment systems. This strong correlation between the two industries will intensify in the near future, and as a result, infotainment systems and in-car experiences as a whole will take massive technological leaps. Innovation is the foundation of technological development. A pivotal innovation in infotainment systems might just be the key to revolutionizing the automotive industry.