University of Colorado
Boulder

ECEN 5763 – Embedded Machine Vision and Intelligent Automation

AUTONOMOUS VEHICLE

BY

AMREETA SENGUPTA

VATSAL SHETH

8/9/2019

University of Colorado Boulder

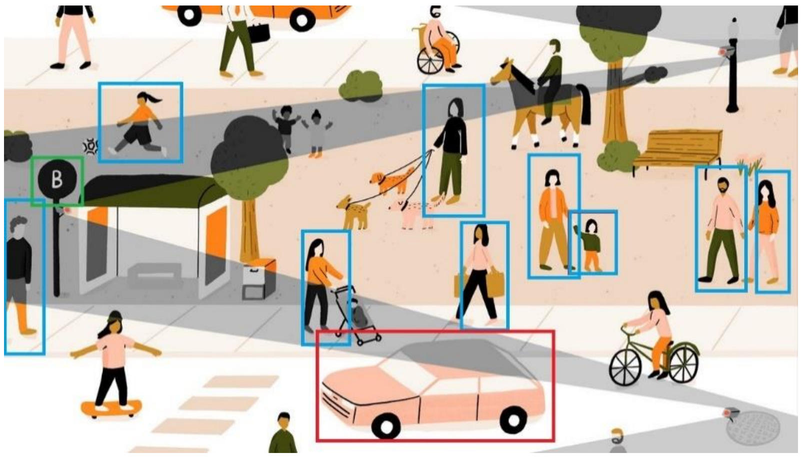
Table of Contents

INTRODUCTION

With the ever-growing advancements in the field of machine vision and machine learning, the goal of implementation of driver-less vehicles are starting to become a reality. In fact, several companies including Mercedes-Benz, General Motors, Bosch, Nissan, Toyota etc. have been developing prototypes of autonomous vehicles.

With this project, we aim to utilize machine vision and machine learning algorithms and techniques to develop and implement few features of an autonomous vehicle with real-time performance. The system developed is capable of steering itself in various obstacle-filled environments without human intervention implemented from the machine vision perspective. This project can contribute to the real-world in the following ways:

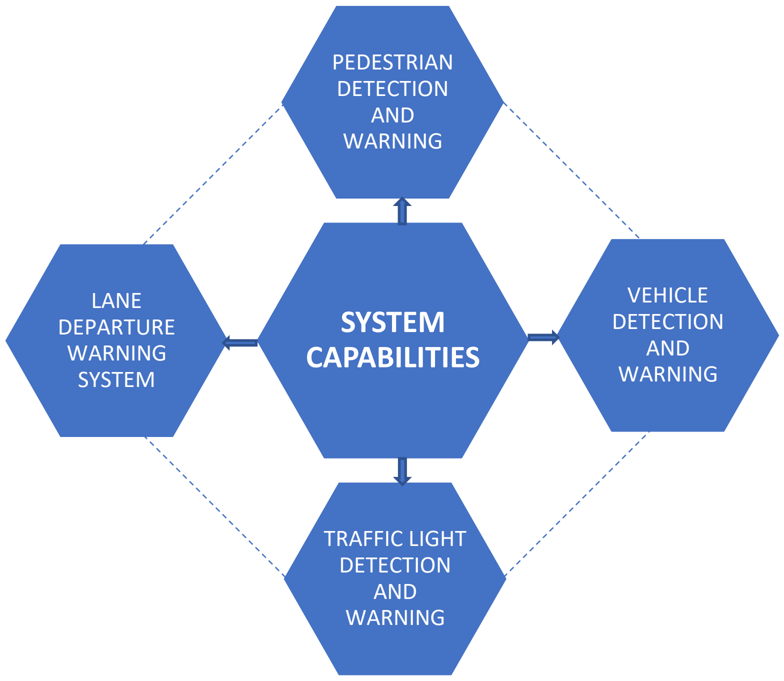
- Millions of people throughout the world die due to motor vehicle accidents. A self-driving vehicle could be a step towards safer transportation options and could be instrumental in reducing the deaths caused by inattentive or distracted drivers.
- A self-driving vehicle could be a very good option for people who are unable to drive or people with disabilities or elderly people, making their commute more comfortable.
- It could also help in traffic management due to automatic road sign detection and regulation following vehicles.



The proposed system will exploit machine vision algorithms to perform lane, car, pedestrian and road sign detection which meet real-time requirements with sufficient accuracy so as to enable the vehicle to navigate in an urban environment with traffic and other obstacles.

The various modules mentioned above are time consuming and hence the system employs a multi-threaded approach in order to execute the image-processing tasks parallelly on separate cores in order to improve the system throughput and performance. The accuracy of the detections depends on the efficiency of the machine learning technique employed.

FUNCTIONAL (CAPABILITY) REQUIREMENTS


PEDESTRIAN DETECTION AND WARNING
LANE DEPARTURE WARNING SYSTEM
SYSTEM CAPABILITIES
VEHICLE DETECTION AND WARNING
TRAFFIC LIGHT DETECTION AND WARNING

- **Lane Departure Warning System:** The system will continuously scan and detect the lane in front of the vehicle and display a warning message on crossing a particular lane. The lane detection algorithm implementation includes creation of the lane detection mask first on a single frame and after continuously capturing the image some pre-processing steps are involved. These include cropping the image frame, converting it to grayscale, blurring the image and extracting a triangular mask of region of interest. Post this, Probabilistic Hough Transform is applied on the image and lines with acceptable slope criteria are filtered. Next, the left and right lanes are separated, and lines will be drawn on each lane enabling the car to navigate safely.

- **Pedestrian Detection and Warning:** The system is designed to detect the pedestrians within a certain range and display a warning message for the same to avoid accidents. The pedestrian detection algorithm implementation includes capturing each image frame and re-sizing them so that a certain number of rows and columns are eliminated. This is done in order to speed-up the detection process. Post this, HOG descriptor along with SVM detector are applied in order to detect pedestrians on each image frame. Lastly, a bounding rectangle is drawn around each detected pedestrian.

❖ **Vehicle Detections and Warning:** The system is designed to detect the cars within a certain range and display a warning message for the same to navigate safely and avoid accidents. The vehicles are detected using the Haar Cascade Algorithm. The car detection algorithm implementation includes loading the dataset for cars. After capturing each image frame, some pre-processing operations are performed like cropping and resizing each image frame. Next, the car is detected using the cascade classifier. Lastly, a bounding rectangle is drawn around each detected car.

❖ **Traffic Sign Detection and Warning:** The system is designed to detect particular road signs like 'YIELD' sign. The road signs are detected using the Haar Cascade Algorithm. The road sign detection algorithm implementation includes loading the dataset for road signs. After capturing each image frame, some pre-processing operations are performed like cropping and resizing each image frame. Next, the road sign is detected using the cascade classifier. Lastly, a bounding rectangle is drawn around each detected sign.

PROJECT PROPOSAL SPECIFICATIONS

❖ **Minimum Goals**

- Perform lane detection with more than 50% accuracy.
- Perform pedestrian detection with more than 50% accuracy.
- Perform car detection with more than 50% accuracy.
- Perform traffic light detection with more than 50% accuracy.

❖ **Target Goals**

- Implement a Lane Departure Warning System.
- Implement pedestrian detection algorithm with an accuracy of more than 70%.
- Implement car detection algorithm with an accuracy of more than 70%.
- Implement traffic light detection algorithm with an accuracy of more than 70%.
- Warning the driver incase pedestrian is detected on the road by annotating the output frames with appropriate message.
- Time measurement and frame rate measurement for real-time performance analysis.

❖ **Optimal Goals**

- Pthread library for multithreading will be used to optimize the performance. There will be 5 threads in total, 4 threads for each feature and one for capturing frame and computing common tasks like converting to gray scale. Inter process communication or signals will be used to schedule feature threads by frame capture thread.
- Lane Detector algorithm achieves optimization by cropping the image to half. As road will always be below the horizon so, it is safe to cut the image into half to save execution time. To minimize the line detection for Hough Transform, Triangular ROI mask will be used. This mask will basically extract only the part of image which has road and neighboring lanes. Slope based filtering will be done to separate left and right lane lines and draw only one line for each side.
- Basic learning of CUDA and try to integrate it in this project.

- Profiling of each task will be done based on time measurements and further scheduling optimizations will be considered if possible.
- Feature detection is most time expensive task. Car, road sign, pedestrian detection is not required to be performed on all frames continuously. Pedestrian will be performed for example once for every 3 frames and road sign and vehicle every 5 frames. This will effectively increase these tasks deadline. Based on profiling then appropriate statics scheduling policy like Rate Monotonic will be chosen.

We were able to achieve all our minimum goals. From the target goals we were able to implement pedestrian detection, car detection, traffic light detection algorithms with more than 70% accuracy. For pedestrian detection, we annotated the video frames with a "Slow Down" warning message. From the optimal goals, we did everything. We tried implementation using CUDA. We could either use NVIDIA CUDA library instead of OpenCV or we could use GPU library of OpenCV which uses implementation based on CUDA. However, detection performance for GPU library was poor as OpenCV has not updated it. NVDIA claimed to give 8x Speedup for cascade detection with the use of GPU library. However, only 2.5x speedup was actually achieved. This could be because CPU to GPU memory transfer of input frame and GPU to CPU memory transfer of output frame are time consuming. Hence, we decided to use multi-threading approach for our implementation.

MACHINE VISION AND MACHINE LEARNING REQUIREMENTS

MACHINE VISION TECHNIQUES AND ALGORITHMS USED:

❖ **Image capturing and encoding**

❖ **Post capture image transformation**
- Resize: For resizing an image frame.
- Cvtcolor: For converting an image to grey scale.

❖ **Image parsing and understanding**
- Canny Edge Detection: For detection of the edges of the objects in the image frame.
- Hough Line Detection: For detection of the lines present in the image frame.
  HOG Descriptor: Feature descriptor which represents the objects as a single feature vector.
- Haar Cascade: Object Detection Algorithm.

MACHINE LEARNING TECHNIQUES AND ALGORITHMS USED:

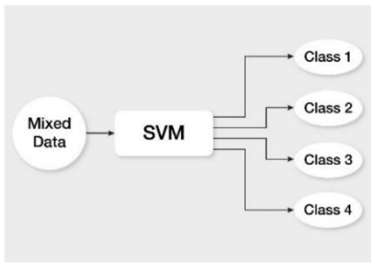
❖ **Generation of XML file for Haar Cascade Classifier**
These are the steps for generating the XML file but due to time constraint we have used XML files sourced from third party contributor, the link for which is available in the references.

- It involves collection of positive (i.e. images containing the object of interest) and negative images (i.e. images without object of interest).

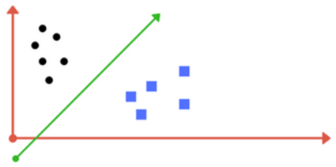
- The positive images are cropped such that only the object of interest is clearly visible.
- The positive and negative images are combined and then run through opencv_createsamples application in order to create samples which can be utilized for training purposes. This is done using a perl script called createtrainsamples.pl.
- A vector file (.vec) is created next based on the samples for a unified training set.
- The opencv_haarcascade application is used to create the haarcascade xml file

❖ <u>**Support Vector Machine (SVM) Classifier**</u>

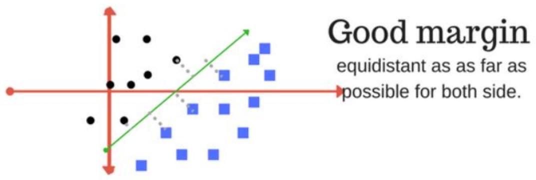
- Support Vector Machine Classifier (SVM) in used in Pedestrian detection in order to detect the pedestrians in the image frames.



- It is a discriminative classifier which is used to classify the detected features. It uses positive and negative examples for training and it then makes a binary decision which is to classify the objects as human or non-human.



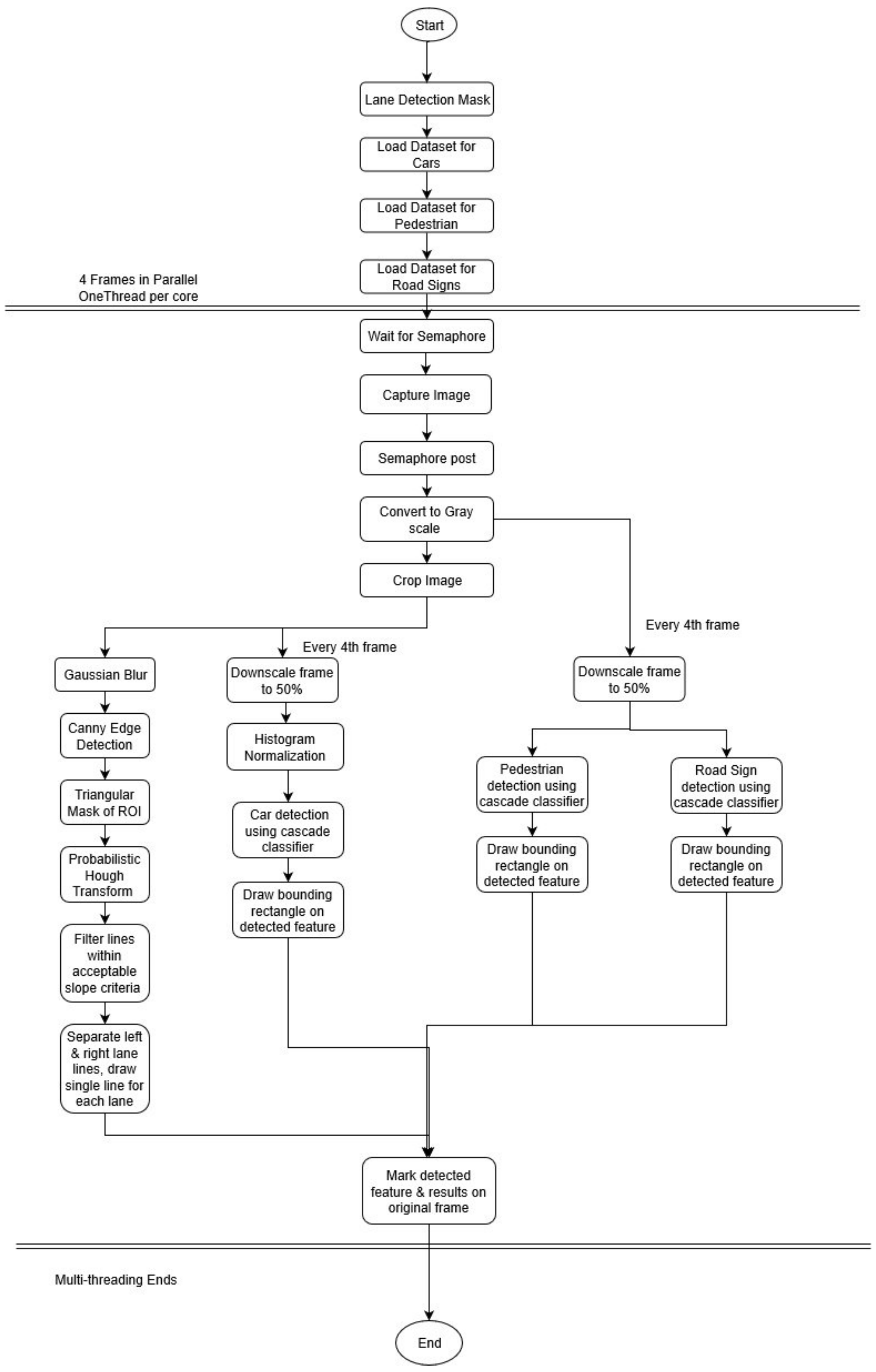
- Its working involves creation of a line or a hyperplane which separates the data into classes (human and non-human in this case) as shown above.



- The support vectors are computed which are the points closest to the created line from both the classes. The distance between the line and the support vectors is called the margin. The margin that has the maximum value determines the optimal hyperplane.

- Thus, SVM tries to make a decision boundary which allows maximum separation between the two classes (i.e. human and non-human).

FUNCTIONAL DESIGN OVERVIEW AND DIAGRAMS

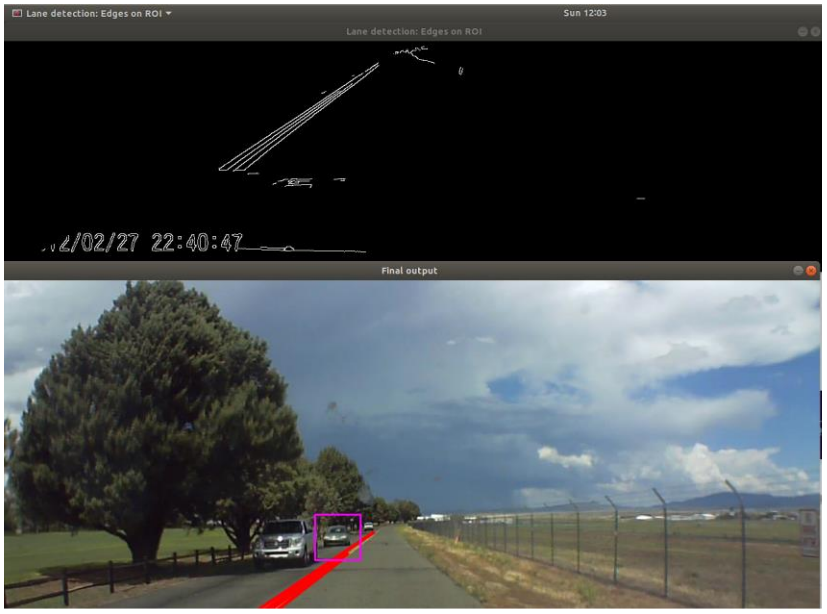

Start
Lane Detection Mask
Load Dataset for Cars
Load Dataset for Pedestrian
Load Dataset for Road Signs
4 Frames in Parallel
OneThread per core
Wait for Semaphore
Capture Image
Semaphore post
Convert to Gray scale
Crop Image
Every 4th frame
Every 4th frame
Gaussian Blur
Canny Edge Detection
Triangular Mask of ROI
Probabilistic Hough Transform
Filter lines within acceptable slope criteria
Separate left & right lane lines, draw single line for each lane
Downscale frame to 50%
Histogram Normalization
Car detection using cascade classifier
Draw bounding rectangle on detected feature
Downscale frame to 50%
Pedestrian detection using cascade classifier
Draw bounding rectangle on detected feature
Road Sign detection using cascade classifier
Draw bounding rectangle on detected feature
Mark detected feature & results on original frame
Multi-threading Ends
End

❖ **Lane Detection**

- As lanes will never go above horizon, so it is safe to crop the image into half. This directly reduces computation time as it is halved.

- Image is converted to gray scale; noise is reduced using Gaussian blur and canny edge detection is used to detect lanes.

- Even in this binary image, our region of interest is just the triangle as seen in below screenshot. This is because we are just interested in lane in which we are driving to warn if we cross that. So, a triangular mask is applied on this edge detected binary image. If we apply this mask first before canny edge detection to save on execution time, then it will detect the edges of triangle itself. To filter that is difficult as road lane lines can also have that similar slope value. Thus, canny edge detection is applied first and then mask is applied to filter out all other detected lines outside our region of interest.



- Probabilistic Hough transform is used to detect lines from the image. Threshold value for this transform is set to 150 as it is a binary image and minimum line length of 100 to filter out small stray lines if it has found somewhere in image.

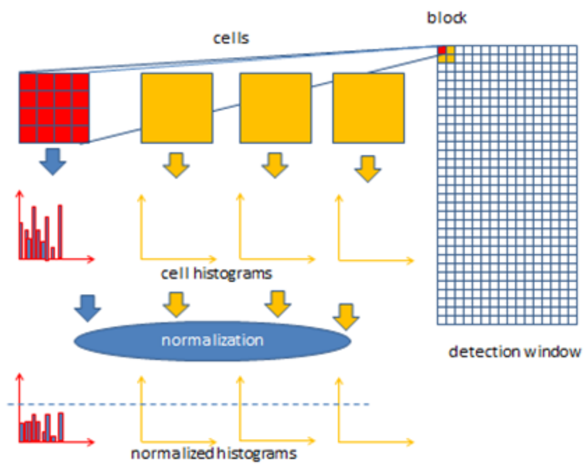
- To extract lane lines from these set of lines, slope is calculated for each line and if its absolute value is greater the 0.5 then only it is considered valid. This step filter outs almost horizontal lines.

- Lines which pass all above criteria is drawn on original frame. Lines need to be drawn at an offset because the line coordinates are as per cropped frame dimensions, but we are drawing it back to original size frame.

❖ **Pedestrian Detection**

- The system employs the HOG algorithm along with SVM to perform pedestrian detection.
  HOG can be used to describe the object of interest in an image using the distribution of intensity gradients. Gradients are employed as the magnitude of gradients have a larger value around the edges or regions with change in intensity which can be used to extract information about the object of interest.

- Essentially, HOG is a feature descriptor which represents the object of interest as a single feature vector. The HOG descriptor for each position in the image is computed by applying a sliding window detector over the entire image which is then combined to give a single feature vector. The HOG descriptor is invariant to image deformation, changes in illumination and contrast making it a robust descriptor.



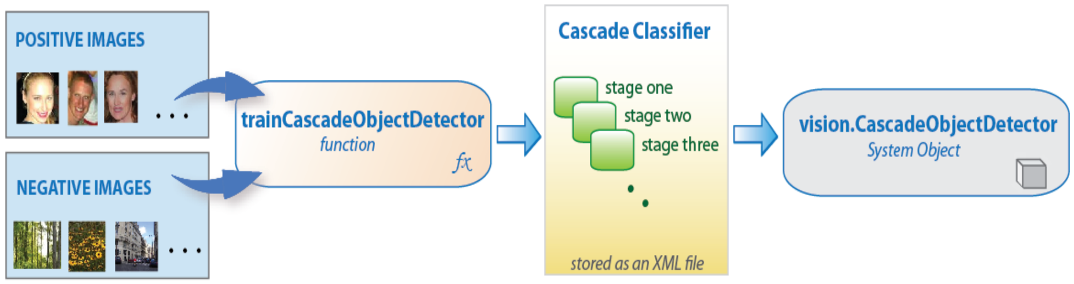
- The pixels of the image are grouped into cells and each of these cells are utilized to extract the histogram feature based on the gradient orientation generating the gradient vector which can be represented as a histogram. The image cells are split into bins and each bin corresponds to a gradient direction. The gradient magnitude is utilized for selection of the weight used for voting into the histogram.

- To make it invariant to changes in illumination, the gradients are normalized using a method called block normalization which normalizes larger segments of the image (i.e. a group of cells) in the form of blocks. Using this, the HOG feature vector is calculated which is basically the concatenation of the histograms, and these features help in extraction of the representations of the objects in the image.

- Support Vector Machine (SVM) is a discriminative classifier which is used to classify the detected features. It uses positive and negative examples for training and it then makes a binary decision which is to classify the objects as human or non-human.

❖ <u>Car Detection and Road-Sign Detection</u>

- The system employs the Haar Cascade algorithm to perform car detection and road sign detection.
- Haar Cascade is basically an object detection algorithm that is employed to detect specified object in an image for which it has been trained. It utilizes machine learning based approach for which it requires a set of positive images i.e. images of object and a set of negative images i.e. images without the object which the cascade function uses to train itself and extract features of the object.



- A Haar feature is extracted by considering the adjacent rectangular regions at a specific location in a detection window, summing up the pixel intensities in each region and calculating the difference between these sums.
- However, among the huge set of the extracted features only a small subset of features is relevant. Hence, to select the useful features the concept of Adaboost is employed, wherein it determines the best threshold which will help in distinguishing between the objects of interest and the non-objects
- During the detection phase, a window of the target size is moved over the input image, and for each subsection of the image, Haar features are calculated.
- For quicker and more accurate detection, a set of Haar features are grouped into a cascade classifier. These cascades of classifiers are applied one by one, where the first few stages will have lesser features and if an image fails this stage, it is discarded and if the images passes this stage, then the next stage is applied to it. Each stage in the cascade should have a low false negative rate for accurate detection. For improved results, the number of stages can be increased, and better-quality images can be applied.
- Once the object of interest is detected, it returns the position of the detected object.

CODE EXPLAINATION AND CHALLENGES FACED

CODE EXPLANATION

- Using image dimension from first frame mask is created for lane detection logic.
- Cascade dataset is loaded in classifier.
- Frame is captured from input video.
- Lane detection:
    - Frame is cropped to half.
    - Converted to gray scale.
    - Canny edge detector is applied.
    - Mask generated above is applied.
    - Probabilistic Hough Lines transform is used to detect lines.
    - Only those lines which has absolute slope value greater than 0.5 are drawn over original frame.
    - All the lines with a slope value greater than 0.5 are stored in a set of Right Lanes.
    - All the lines with a slope value lesser than -0.5 are stored in a set of Left Lanes.
    - For each set, the extreme coordinates among all the lines are determined. This helps in reducing the set of lines to a single line.
    - Since the frame is cropped into half horizontally before lane detection to reduce the computation time, the image frame is again returned to its original size by adding half the image height to the x coordinate.
    - The extreme two points determined are utilized to mark the line on the lane.
    - Lane Departure Warning System to be implemented.
- Car detection:
    - Image frame is converted to grey scale.
    - The image frame is cropped to half horizontally. This is done with the assumption that car won't be above the horizon and even if is it, it is too far and not consequential in the car navigation.
    - Next, the image frame is resized and downscaled by 50%. This is done because Haar Cascade implementation is time consuming and this helps in saving time.
    - The XML file provided online along with the detectMultiScale API is used to detect the car. The argument for minimum feature size is given as 30X30 so that cars below this size can be ignored in order to save time. Even a very small car will have a size equal to or greater than 30X30 and if there is a car smaller than this size, then it is inconsequential as it is too far away.
    - Since the image is downsized before, the x, y, height and width coordinates are multiplied by two to get back the original size.
    - Since the frame is cropped into half horizontally before car detection to reduce the computation time, the image frame is again returned to its original size by adding half the image height to the x coordinate.
    - Bounding rectangles are drawn around the detected cars on the image frames.

- Pedestrian Detection:
    - Image frame is converted to grey scale.
    - Next, the image frame is resized and downscaled by 50%. This is done because HOG Descriptor and SVM Classifier implementation is time consuming and this helps in saving time.
    - Next, the HOG descriptor along with SVM detector is applied on each image frame to detect the pedestrians.
    - Since the image is downsized before, the x, y, height and width coordinates are multiplied by two to get back the original size.
    - A bounding rectangle is drawn around each detected pedestrian on the image frame.
- Road Sign/Traffic Light Detection:
    - This module is yet to be implemented.
- Multi-threading:
    - The workflow of the designed system is such that the tasks common to all the detections is done first and then for the tasks specific to each detection, it branches to perform the different tasks.
    - As seen from the block diagram above, four threads have been used and each of these threads will do the detections in parallel.
    - This concept is similar to the pipelining in computer architecture. In any case i.e. if the frame is captured from a video or a camera, the initial time to fill the pipeline, i.e. time taken for all the four cores to start working, is the only time when the CPU is idle.
    - Once all the cores have started working, ideally it should give a throughput of 1 frame per cycle.
    - Capturing of frame is protected using semaphore in order to prevent simultaneous frame access from multiple threads.
    - Since car detection, pedestrian detection and road sign detection are execution intensive and time-consuming tasks, a safe assumption is made that there won't be any significant change in features in the successive frames, and hence these detections are performed on every 4th frame.

CHALLENGES FACED

- **Feature Detection**
    - Tuning performance of cascades: Due to time constraints, we were not able to train the cascade classifier with respect to our input video, hence it results in false positive and false negative detections.
    - Grouping: Multiple detections on a single feature i.e. more than one bounding rectangle for one single feature.
    - Partial Feature Detection: Features get detected but the bounding rectangle does not cover the entire feature. For example, if a car gets detected but the bounding rectangle only covers the windows of the car.
- **Lane Detection**
    - Difference between left and right side of lane based on slopes.

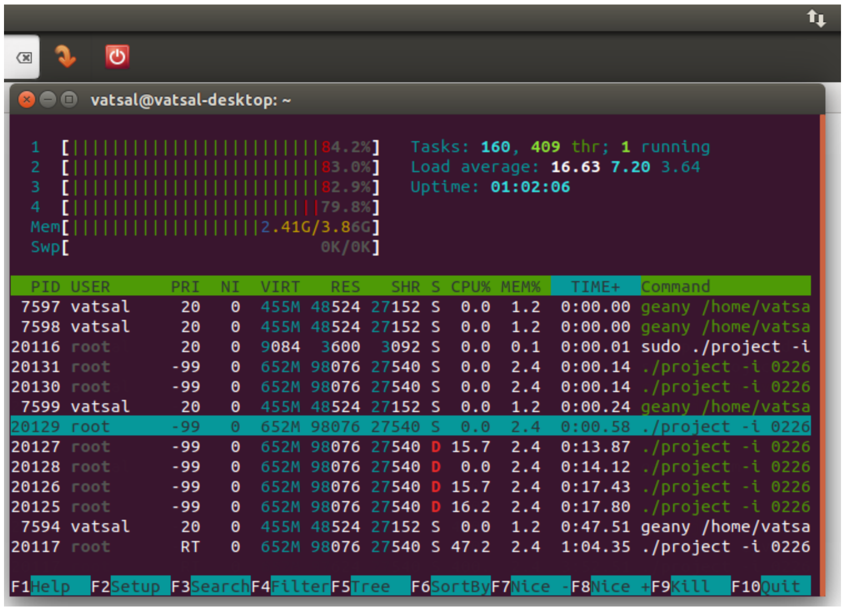
- Grouping multiple line detected for each lane into one.
- Maintaining the balance of minimum line length. For example, if the line length is too small, it may detect stray lines from the frame which are not actual lane marking and if it is too big, it might not detect the dashed lane lines.

❖ **Multi-threading**

- Developing a logic that will achieve maximum CPU utilization while making sure that it doesn't result in deadlock in the worst-case situation.

SYSTEM PERFORMANCE ANALYSIS

CPU Utilization



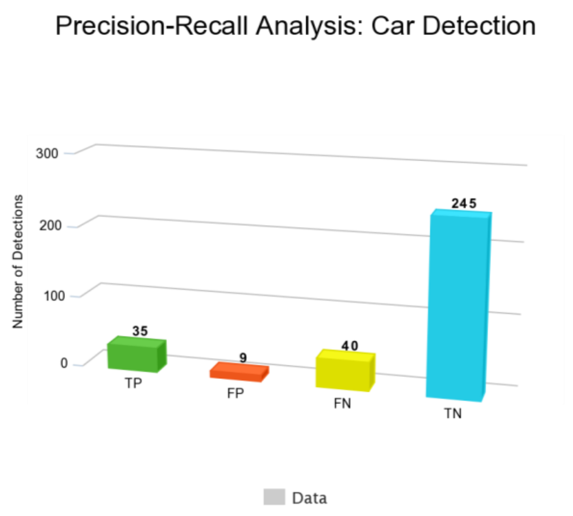
```
vatsal@vatsal-desktop: ~

  1  [||||||||||||||||||||||||84.2%]   Tasks: 160, 409 thr; 1 running
  2  [||||||||||||||||||||||||83.0%]   Load average: 16.63 7.20 3.64
  3  [||||||||||||||||||||||||82.9%]   Uptime: 01:02:06
  4  [||||||||||||||||||||||||79.8%]
  Mem[||||||||||||||||||2.41G/3.86G]
  Swp[                         0K/0K]

  PID USER       PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 7597 vatsal      20   0  455M 48524 27152 S  0.0  1.2  0:00.00 geany /home/vatsa
 7598 vatsal      20   0  455M 48524 27152 S  0.0  1.2  0:00.00 geany /home/vatsa
20116 root        20   0  9084  3600  3092 S  0.0  0.1  0:00.01 sudo ./project -i
20131 root       -99   0  652M 98076 27540 S  0.0  2.4  0:00.14 ./project -i 0226
20130 root       -99   0  652M 98076 27540 S  0.0  2.4  0:00.14 ./project -i 0226
 7599 vatsal      20   0  455M 48524 27152 S  0.0  1.2  0:00.24 geany /home/vatsa
20129 root       -99   0  652M 98076 27540 S  0.0  2.4  0:00.58 ./project -i 0226
20127 root       -99   0  652M 98076 27540 D 15.7  2.4  0:13.87 ./project -i 0226
20128 root       -99   0  652M 98076 27540 D  0.0  2.4  0:14.12 ./project -i 0226
20126 root       -99   0  652M 98076 27540 D 15.7  2.4  0:17.43 ./project -i 0226
20125 root       -99   0  652M 98076 27540 D 16.2  2.4  0:17.80 ./project -i 0226
 7594 vatsal      20   0  455M 48524 27152 S  0.0  1.2  0:47.51 geany /home/vatsa
20117 root        RT   0  652M 98076 27540 S 47.2  2.4  1:04.35 ./project -i 0226

F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

- The htop command has been used to check the CPU Utilization.
- As seen above, all the four cores are running at around 80% equal utilization which validates our concept explained under the Multi-threading topic in Code Explanation.
- Linux has a non-preemptible kernel which makes it unsuitable for real-time applications. Other processes related to kernel and other running applications depending on their resource requirements directly affects our application. This thing is difficult to model but can be accounted for by having a safety margin which in this case is around 20%.

University of Colorado
Boulder

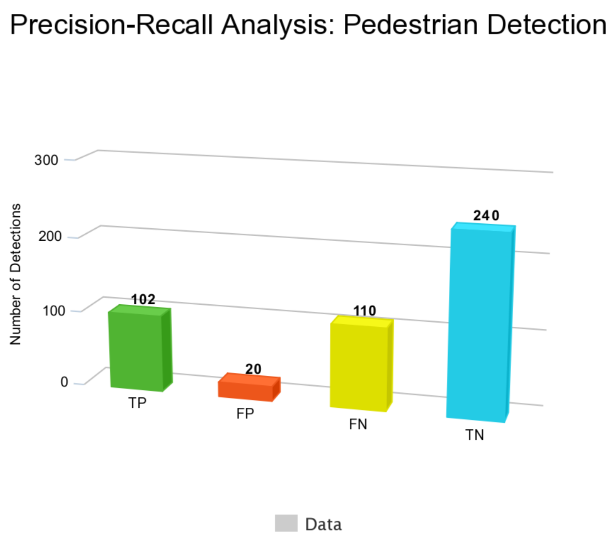
Precision-Recall Analysis

❖ Car detection



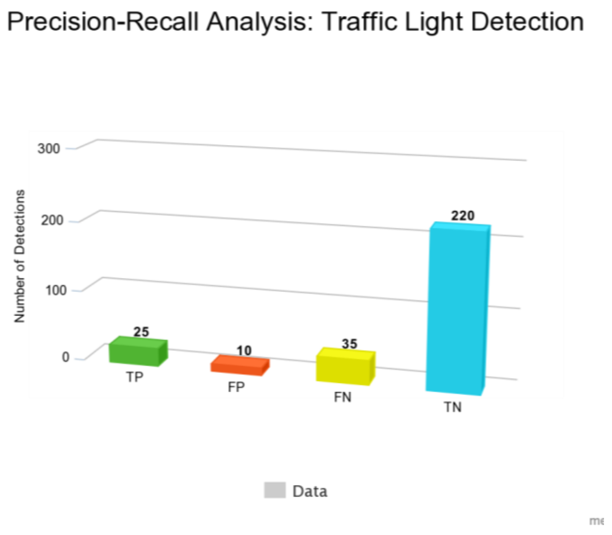
- $Accuracy = \frac{TP+TN}{Total} = 0.85$
- $Precision = \frac{TP}{TP+FP} = 0.79$
- $Recall = \frac{TP}{TP+FN} = 0.46$
- $F1 = 2 * \frac{precision*recall}{precision+recall} = 0.58$

❖ Pedestrian Detection



- $Accuracy = \frac{TP+TN}{Total} = 0.72$
- $Precision = \frac{TP}{TP+FP} = 0.83$
- $Recall = \frac{TP}{TP+FN} = 0.48$
- $F1 = 2 * \frac{precision*recall}{precision+recall} = 0.61$

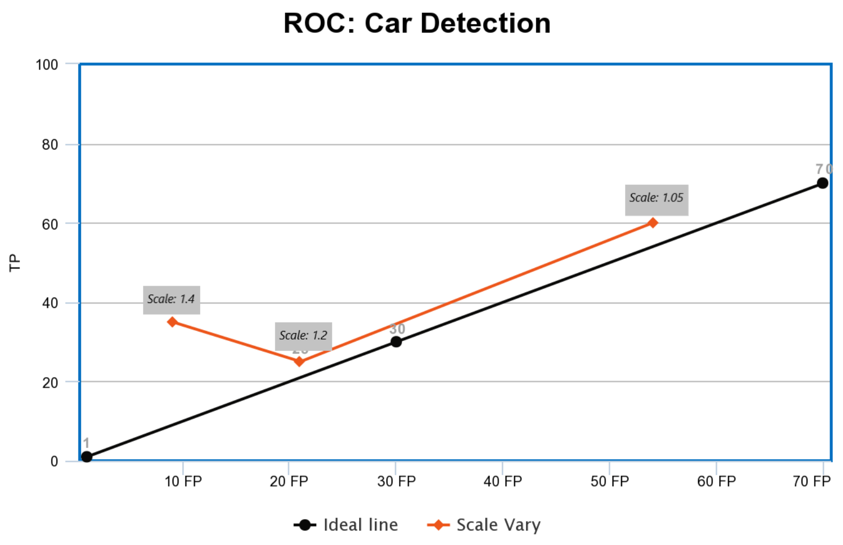
❖ Traffic Sign Detection



- $Accuracy = \frac{TP+TN}{Total} = 0.82$
- $Precision = \frac{TP}{TP+FP} = 0.71$
- $Recall = \frac{TP}{TP+FN} = 0.41$
- $F1 = 2 * \frac{precision*recall}{precision+recall} = 0.52$

ROC Analysis

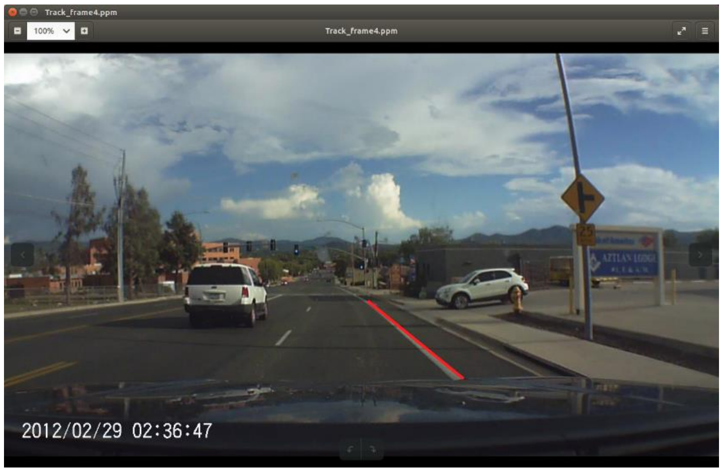
❖ Car Detection



- This curves shows variation in TP and FP values with changing scale parameter in detectMultiScale() API
- This parameter defines the step size for scale pyramid in sliding window Haar cascade detector

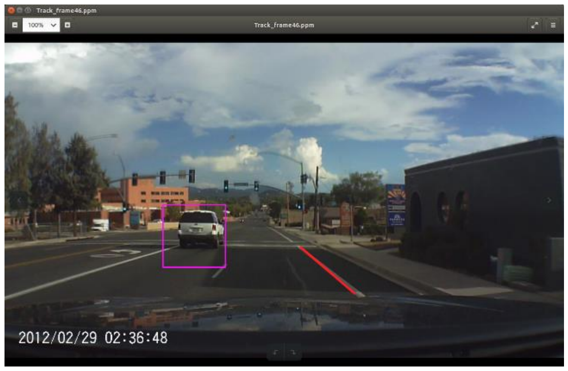
General Observations

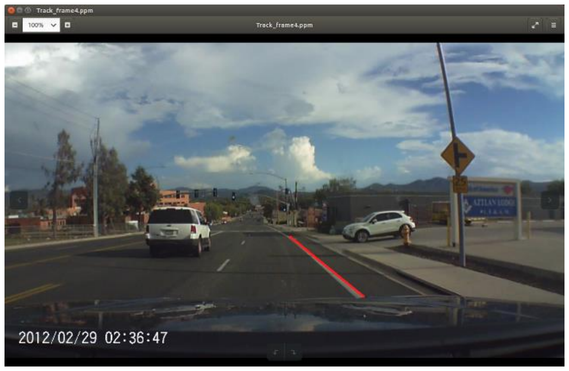
❖ Lane Detection



- The logic to group lines representing same lane has some flaws but the performance is acceptable as per our application requirements. As seen above, the line marking the lane is connecting the two extreme sides of the detected lane. Lane can be considered as a rectangle if we take into account the edges of the solid lane. As we know that two extreme points of a rectangle are diagonally opposite, so our logic will draw a line which basically will be diagonal of this rectangle.
- Our focus should only be on the lane in which our car is driving as per the autonomous vehicle application. Thus, the triangular mask is working correctly, and we are able to filter out the lines for the adjacent lane.
- Thresholding for edge detection needs to be improved so that the dashed line is also marked as a detected lane.

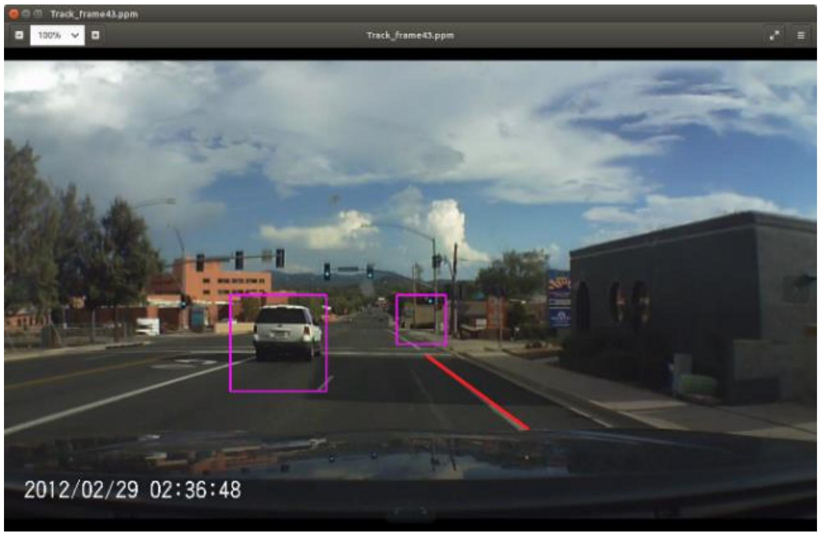
❖ Vehicle Detection



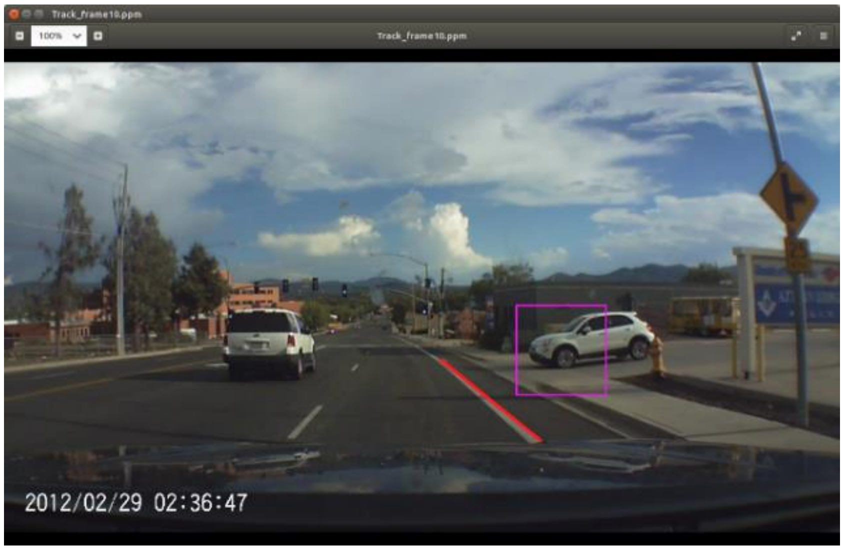


- As seen above, in one frame car is getting detected and in the other frame car is not getting detected.
- Lane is getting detected accurately.

University of Colorado Boulder



In this frame, it can be seen that the car is falsely getting detected.



In this frame, it can be seen that the bounding rectangle is only covering the car

FPS and real-time analysis

❖ Half HD Video [1280X640]

Implementation Approach	**FPS**
Without Frame Skipping	5.08
With Frame Skipping	12.46

- Since there isn't much difference between successive frames, we can skip few frames in order to reduce the processing time and to increase the FPS. As seen above Frame skipping significantly increases the FPS.

❖ FPS for Individual Detections

Detections	**FPS**	
	WITHOUT FRAME SKIPPING	WITH FRAME SKIPPING
Lane Detection	11.20	14.14
Pedestrian Detection	4.78	12.87
Traffic Light Detection	8.18	16.05
Car Detection	9.10	15.49

- As seen above, Lane detection has the maximum FPS as it does not involve feature detection.
- Pedestrian Detection has lower FPS as compared to the other detections as we have used 5% scaling factor for sufficiently accurate detection.
- Traffic Light Detection and Car Detection have higher FPS as we have using 20% scaling factor to perform the detection.

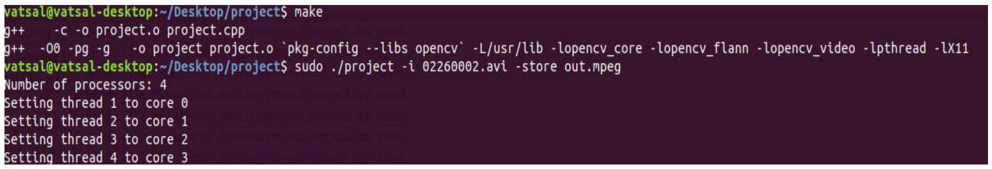
❖ Video [640X480]

Implementation Approach	**FPS**
Without Frame Skipping	16.67
With Frame Skipping	33.84

- As seen above, if we reduce the resolution of the video, the FPS significantly increases. This is because a better-quality video will have more pixels and will take higher processing time and hence will have a lower FPS.

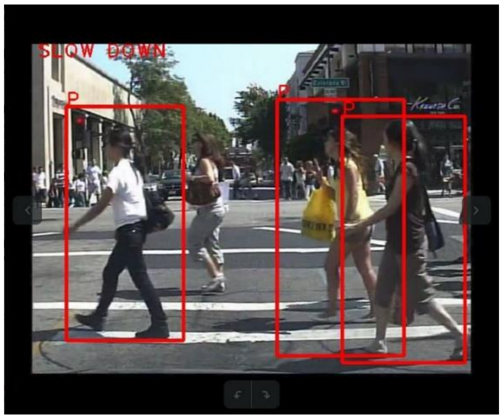
PROOF-OF-CONCEPT

Build and Run



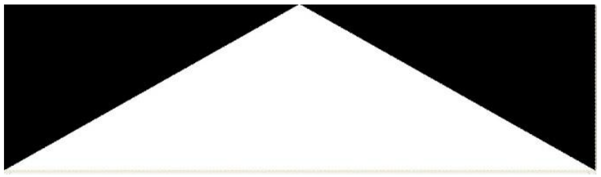
```
vatsal@vatsal-desktop:~/Desktop/project$ make
g++    -c -o project.o project.cpp
g++  -O0 -pg -g   -o project project.o `pkg-config --libs opencv` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video -lpthread -lX11
vatsal@vatsal-desktop:~/Desktop/project$ sudo ./project -i 02260002.avi -store out.mpeg
Number of processors: 4
Setting thread 1 to core 0
Setting thread 2 to core 1
Setting thread 3 to core 2
Setting thread 4 to core 3
```

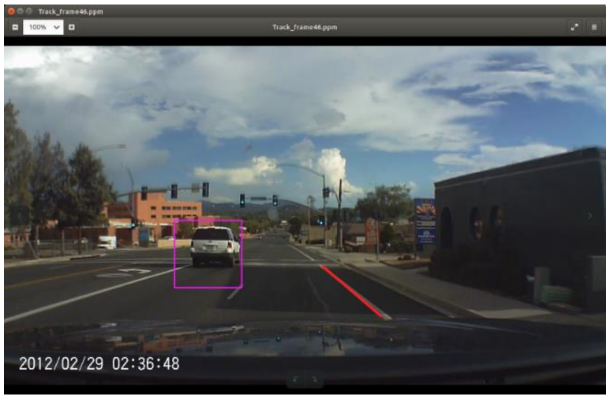
Pedestrian Detection



- As seen above, Pedestrians are getting detected and Slow Down warning is annotated on frame.

Lane Detection and Car Detection





- As we can see, lane is getting detected on the right.
- However, dashed lane lines are not getting detected.
- Concept of triangular mask is working correctly, marking only the lanes in which the car is currently driving and not the one beside it.
- Car is getting completely detected, as seen above.
- Final annotated video is available on GitHub.

CONCLUSION

This project has been implemented using machine vision and machine learning concepts. It promotes elimination of human intervention by facilitating self-driving cars. The modules have been implemented using OpenCV and include lane detection and warning system, pedestrian detection, car detection and road sign detection. In terms of real-time, implementation of an autonomous car on a large scale is quite challenging. However, we have used the machine vision concepts that we have learned to design few features of this system for a safe and reliable implementation of the same. The project has helped us translate the machine vision concepts that we learned like image capturing, image encoding, image transformation, image parsing etc. into a real time application. The various modules have been integrated using a multi-threaded approach on separate CPUs to meet the real time requirements of the system. The accuracy of the system can be improved by improving the efficiency of the machine learning algorithm employed with better training sets. Overall, this project helped us build an efficient real-time system for accurate obstacle detection and navigation in any environment. The future scope of this system can include control of the car accelerator, break, steering and use of Google Map APIs for navigation control.

ACKNOWLEDGEMENTS

We would like to thank Professor Sam Siewert for his guidance and for providing us the with the opportunity to work on a very relevant real time project which allowed us to translate our knowledge into a real-time application.

We would also like to express our gratitude to the TAs for their constant help and constructive suggestions for the implementation of our project.

REFERENCES

- https://www.cargroup.org/autonomous-vehicles-bumps-on-the-road-to-our-smart-mobility-future/
- https://www.curbed.com/2018/1/17/16897222/machine-learning-urban-planning-sidewalk-labs
- https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-gradients-hog-descriptor
- https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989
- http://www.willberger.org/cascade-haar-explained/
- https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72
- http://www.cse.uaa.alaska.edu/~ssiewert/a485_code/
- https://docs.opencv.org
- Embedded Machine Vision Exercise 1 by Amreeta Sengupta
- Embedded Machine Vision Exercise 5 by Amreeta Sengupta
- Embedded Machine Vision Exercise 5 by Vatsal Sheth

APPENDICES [Available in GitHub]

1. Code
2. Supporting Materials [Video (.mpeg files) and XML files]