Self-Driving Auto-Pilot Project Report
ECEN 5763 – Embedded Machine Vision and Intelligent Automation

Avinash Dhulehole & Vijoy Sunil Kumar
Summer 2017

Index

Introduction

A self-driving vehicle is one that is capable of sensing its environment, understanding them and making the right decision to safely navigate from point A to point B without human inputs. Self-driving vehicles uses Machine Vision to interpret or identify mainly navigation paths, obstacles as well as road signs from variety of sensors such as RADAR, LIDAR, GPS, Visible, IR cameras etc.

A self-driving car project seemed more apt than others because it involves the application of almost all of the algorithms and concepts that was taught in the ECEN 5763 class and would be great way to apply it in practice. My proposal for the final project would such a system where the processing is carried out solely on the data from the video recorded from cameras.

Most self-driving vehicles, including Google's Waymo are loaded with sensors, including cameras, ultrasound, high-accuracy GPS, and expensive laser-ranging instruments known as lidar.
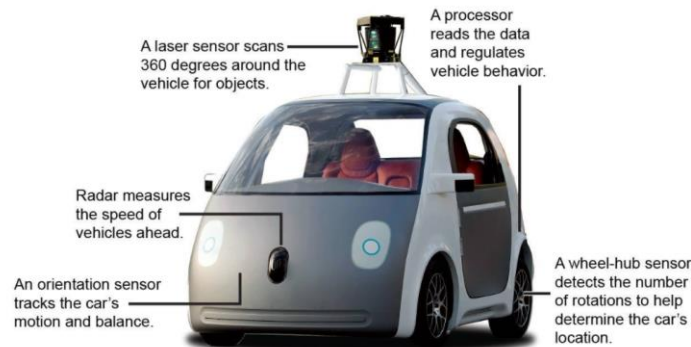

Fig: Google's self-driving car "Waymo"

All these sensors together help the car build up a composite picture of the surrounding world in order to drive safely. But some components, such as the lidar are really expensive costing about tens of thousands of dollars.

Self-driving car project was chosen because it is going to be the next big thing and almost all major car manufacturers are currently working on some or the other form of automation. The challenges of a self-driving vehicle project would be to accurately detect, recognize and classify an obstacle and take appropriate action in real time.

This project would make use of pre-recorded video data to analyze the scenes and take necessary actions which brings us to the next major challenge. Since we are not using RADARs or LIDARs, decisions would completely depend upon the camera feed making it less dependable. But the biggest advantages are hardware cost and integration. Cameras are much cheaper than radar antennas and lidar sensors. The disadvantage is that they are collecting less data on which their system can make decisions on.

Although not in the recent future, an autonomous car solely depending on vision based system would become a reality not just to bring down the cost but because it is possible.

Finally, an evaluation was done to assess the performance of the program in implementing the features listed in the next section.

Features

The following features are implemented in this project to enable a self-driving auto-pilot to take decisions to safely navigate from source to destination.

Feature #1 – Lane detection

Lane detection is a basic feature in all self-driving cars and is used to position the vehicle relative to the road lanes. Without lane detection, a self-driving car would not know where to position itself.

This project implements lane detection based on Hough line transform and its working is based on certain constraints.

- Constraint (1): Reliable lane detection is possible only on a well-lit day since the recorded video used here was during the day and the algorithm was developed solely on this lighting condition.
- Constraint (2): Lane detection does not work on parking lots or places where lanes are not drawn.

Feature #2 – Lane departure hazard

This feature is part of the hazard detection system implemented in the project. A warning logo pops up on the video feed whenever the vehicle changes lane. More information on different warning logos can be found in the *Working* section. The lane departure warning system is based on the same constraints as of lane detection system listed above.

Feature #3 – Vehicle detection

Vehicle detection feature is a part of the obstacle detection system used in almost all self-driving cars. Auto manufacturers used RADAR or LIDAR along with camera data for reliable obstacle detection, however this project implements the system using a single visible camera and a trained data set of 526 car images using Haar Cascades. A RED bounding box will be drawn over the detected vehicle. The implementation is based on the constraints listed below

- Constraint (1): Vehicle detection region of interest will be limited to only those vehicles on the front and in the same lane.
- Constraint (2): Reliable vehicle detection is possible only on a well-lit day since the recorded video used here was during the day and the algorithm was developed solely on this lighting condition.

Feature #4 – Vehicle distance

Vehicle distance estimation is better done using RADAR or LIDARs or even using stereopsis with multiple cameras, however this project implements the system with a single front facing camera. Using the properties of similar triangles, with focal length of camera and camera position height known, and obtaining pixel distance of bottom of car from horizon, the vehicle distance is calculated. The constraints for this implementation are shown below. The distance calculated will be displayed on top of the vehicle detected bounding box in cyan color.

- Constraint (1): The distance calculated will only be an approximate value.
- Constraint (2): Reliable vehicle distance measurement is possible only on active vehicle detection system (*see vehicle detection constraints*).

Feature #5 – Vehicle at close proximity hazard

This feature is part of the hazard detection system implemented in the project. A warning logo pops up on the video feed whenever the vehicle in front is closer than a set limit (25m in this case). More information on different warning logos can be found in the *Working* section.

Feature #6 – Pedestrian detection

Pedestrian detection is an important feature in any self-driving vehicle, since the autonomous system must not only protect the passengers inside the vehicle but also protect the people on the road. Our implementation of pedestrian detection is based on OpenCV's pre-trained HOG + Linear SVM model. The output can then be ultimately used to stop the vehicle at a safe distance in case of an emergency. This project draws a BLUE bounding box over the people detected in front of the vehicle.

- Constraint (1): Detection range will only be limited to pedestrians crossing the road and not those on the sidewalk.
- Constraint (2): Reliable pedestrian detection is possible only on a well-lit day since the recorded video used here was during the day and the algorithm was developed solely on this lighting condition.

Feature #7 – Pedestrian detection hazard

The pedestrian detection hazard is also part of the hazard detection system implemented in the project along with lane departure warning and RED light hazard (feature #8). A warning logo pops up on the video feed whenever a pedestrian is detected in front. The pedestrian detection warning system is based on the same constraints as of pedestrian detection system listed above.

Feature #8 – Traffic lights recognition

This feature is another important and necessary aspect in a self-driving car. The vehicle should identify the status of the traffic light and issue a warning and stop accordingly. This project recognizes the status of the traffic lights and outputs the result on to the video feed.

- Constraint (1): Reliable traffic light recognition is possible only on a well-lit day since the recorded video used here was during the day and the algorithm was developed solely on this lighting condition.
- Constraint (2): Yellow and Red lights will be detected as Red which will then be used to display the warning.

Feature #9 – RED traffic light hazard

This feature belongs to the hazard detection system implemented in this project. A warning logo pops up on the video feed whenever a Red/Yellow light is recognized. The RED light warning system is based on the same constraints as of traffic light detection system listed above.

Feature #10 – Yellow road sign detection

Road sign detection is another important feature in any autonomous vehicle. A YELLOW bounding box will be drawn over the detected yellow road sign. Our project implements this feature with the following constraints.

- Constraint (1): Detected road signs will only include warning (yellow) signs.
- Constraint (2): Reliable detection is possible only on a well-lit day since the recorded video used here was during the day and the algorithm was developed solely on this lighting condition.

Feature #11 – Yellow road sign hazard

The road sign hazard feature completes the hazard detection system implemented in this project. A warning logo pops up on the video feed whenever any of the yellow road sign is detected. The road sign hazard system is based on the same constraints as listed above.

Development Environment

The project is developed and tested on Ubuntu 14.04 with OpenCV version 2.4.8

The below diagram shows the folder structure of the project.

/auto-pilot
- -Makefile
- -auto-pilot.cpp
- -lane_ops.cpp
- -lane_ops.h
- -signal_ops.cpp
- -signal_ops.h
- -traffic_ops.cpp
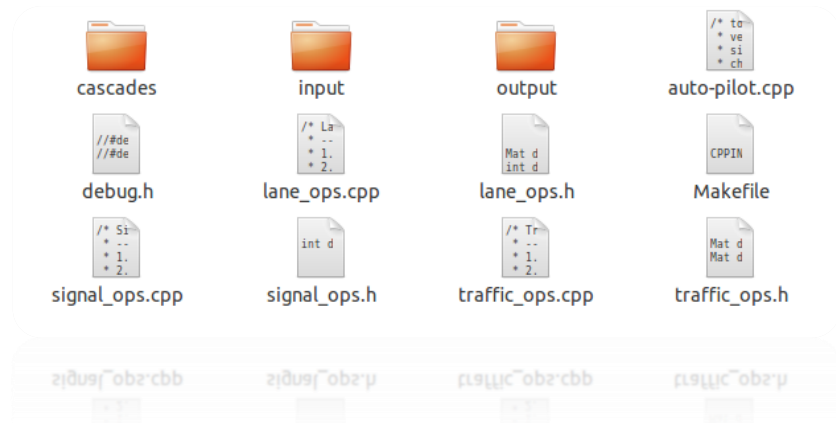- -traffic_ops.h
- -debug.h
- **/input**
  - -front.AVI
- **/output**
  - -output.avi
- **/cascades**
  - -cars.xml

The project is mainly divided into 3 sets of operations namely
1. Lane operations,
2. Signal operations, and
3. Traffic operations

Lane operations are contained in files **lane_ops.cpp and lane_ops.h** which include lane detection and lane departure warning. Traffic lights recognition and road sign detection belong to signal operations which are included in **signal_ops.c and signal_ops.h**. Finally, traffic operations include vehicle detection, vehicle distance estimation and pedestrian detection which are included in **traffic_ops.cpp and traffic_ops.h**

The **cascades** folder contains the Haar cascades training set file "cars.xml" which is used for vehicle detection. The xml file is read in traffic_ops.cpp.

The **input** folder contains the recorded video from the front facing camera which is used as the source video on which all processing is done. The **output** folder will contain the final output video based on the format that is specified in the command line argument "--store"

auto-pilot.cpp contains the main loop where all the function calls to process the different operations are called. Individual operations as well as all the features combined can be called from the main loop using the command line argument "--feature".

debug.h contains certain macros which when uncommented can be used for debugging the application by printing out data into the terminal.

How to Run?

This project was run and tested on Ubuntu 14.04 using OpenCV 2.4.8. You can run the application by following the below steps:

1. Extract the project folder into your desired directory.
2. Open the terminal and navigate into the project directory.
3. Build the program using "make".

```
vijoy@VIJOY-PC:~/Desktop/auto-pilot$ make
g++  -Wall    -c -o auto-pilot.o auto-pilot.cpp
g++ -Wall  auto-pilot.o lane_ops.o signal_ops.o traffic_ops.o debug.h -o auto-pilot `pkg-config --libs opencv` -L/usr/
local/opencv/lib -lopencv_core -lopencv_flann -lopencv_video
vijoy@VIJOY-PC:~/Desktop/auto-pilot$ ./auto-pilot
```

4. Run the program by typing in
   >> ./auto-pilot   input/front.AVI   --feature 0   --store out.avi
   This will run the program by taking in front.AVI, from the input directory, as the source video and enable all features by default. The final output video "out.avi" will be stored in the output directory. You can also enable individual features by specifying the feature number according to the help screen shown in the following steps.

5. You can access the help function by just typing
   >> ./auto-pilot
6. Below is a screenshot of the help function

```
...................HELP.......................
./auto-pilot <VIDEO-NAME.FORMAT> --feature <feature_num> --store <OUTPUT VIDEO FILENAME>

.................Feature_num options...........
0. All features enable
1. Lane detection
2. Lane departure warning
3. Vehicle detection + distance estimation
4. Pedestrian detection
5. Traffic lights recognition
6. Road sign (Y) detection

..................EXAMPLE usage..............
./auto-pilot front.AVI --feature 6 --store out.avi
Output video is stored in output directory

Press ECS to quit OR p/P to pause and r/R to resume
.........................................
```

7. You can exit the program by pressing 'ESC' or pause the program by pressing 'p' or 'P' and resume playing by pressing 'r' or 'R'. (The pause/resume feature is only available on - -feature 0).
8. You can also seek forward or backward using the trackbar in the "source video" window.
9. The program exits automatically at the end of the video.

Working

1. <u>Lane detection</u>

The lane detection algorithm is based on hough line transform and is carried out as shown in the below steps:

    a. Load BGR image from the source video into **src.**
    b. Convert the BGR image to grayscale image **gray.**
    c. Create a mask image **mask** with 2 polygons covering the region of interest for lane detection. **mask** is shown below
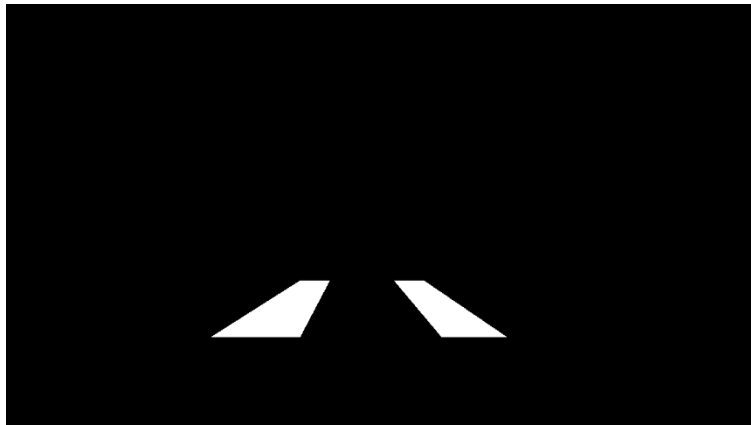
    d. Bitwise AND **gray** with **mask** to produce **masked_image** which is shown below
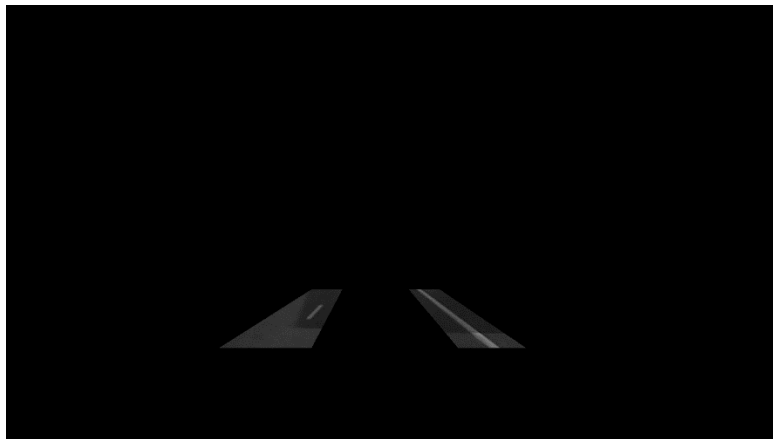
    e. Create two rectangular regions of interest on the BGR image **src** (shown below) and calculate the average pixel intensity in both of these regions (roi_h and roi_l as shown below)

f. Calculate the average pixel intensity difference.
g. Convert **masked image** to binary image **im_white** with the threshold value based on the difference calculated above. In shirt, the thresholding is adaptive in the sense that it varies based on the average pixel intensity in the regions of interest. The binary image will now have the lanes as white and remaining background as black.
h. Apply canny transform on **im_white** to detect edges on the binary image.
i. Then, apply probabilistic hough transform on the edge detected image which will return a set of (x,y) points.
j. Draw lines joining these (x,y) points. The resulting image containing the lines is bitwise OR'ed with the BGR image and displayed.

2. <u>Lane departure warning</u>
    a. Load BGR image from the source video into **src**
    b. Create four rectangular regions of interest on the BGR image **src** (shown below) and calculate the average pixel intensity in all of these regions (roi_up, roi_down, roi_left and roi_right as shown below)

    c. Calculate the average pixel intensity difference in the horizontal direction and in the vertical direction.

    d. It has been found that when the vehicle crosses a lane, the difference between the vertical and horizontal average pixel intensity difference goes above a certain threshold (25 when tested). If it goes above the threshold then a flag **hazard_lane** is set indicating a lane departure.

    e. This flag is then used to display the hazard logo and the type of hazard on the output video.

    f. The hazard logo and message for lane departure is shown below. The '**L**' indicates lane departure.

3. <u>Vehicle detection and distance estimation</u>
   Calculation is done using position information of vehicle in the image. We have to know few definitions to start with the algorithm.
   Bottom line: Contacting line between bottom of vehicle and road surface.
   Horizon: Horizontal line passing through vanishing point of road lanes. Distance between bottom of vehicle to horizon is inversely proportional to the distance to the vehicle. Horizon position in image keeps changing due to vehicle motion and road inclination.

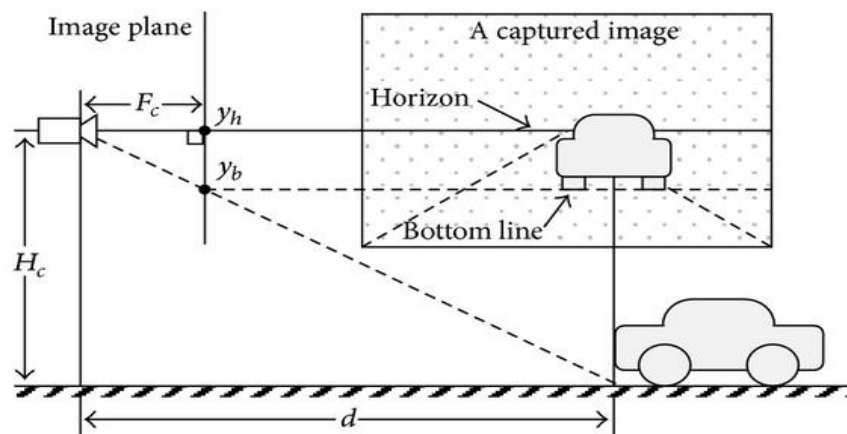   Mathematics behind distance calculation: Similar Triangles.

Figure: Mathematics of vehicle distance calculation

By applying properties of similar triangles to the above picture, d, the distance of vehicle can be calculated using below formula;

$d/F_c = H_c/(y_b-y_h)$

$=> d = F_c * H_c /(y_b-y_h)$

where: F_c = focal length of camera,

H_c = camera height,

y_b = vertical coordinate of vehicle bottom line, and

y_h = vertical coordinate of horizon.

Assumption made: Horizon will pass through center of image when optical axis of camera is parallel to road surface. So, it is assumed that the camera's optical axis used here parallel to road surface.

Algorithm:
a. Capture a frame (RGB color space) from video file loaded.
b. As horizon passes through the middle of the image, vehicles are detected below the horizon. So, a ROI is created in lower half of the frame and such that vehicles in the lane we are currently are detected.
c. Using Haar features for car detection, cars in the ROI are detected and bounding box are drawn for each car detected.
d. Bottom line vertical coordinate, y_b, is obtained from the lower line drawn in the bounding box. Horizon vertical coordinate, y_h, is the vertical coordinate of the line passing through the middle of the original frame. Focal length, F_c, and camera height, H_c, are constant characteristics of camera and its position. Applying this to distance calculation formula, d is obtained.
e. Vehicle distance, *d*, is displayed above the bounding box drawn over the vehicles detected.

4. Vehicle close proximity warning
    a. The vehicle distance function mentioned above sets the flag for hazard detection in this case.
    b. When the distance of the vehicle in front is less than 25 meters a flag **hazard_vehicle** is set.
    c. This flag is then used to display the hazard logo and the type of hazard on the output video.
    d. The hazard logo and message for vehicle at close proximity is shown below. The '**V**' indicates vehicle in front is too close.

5. Pedestrian detection
    a. Load BGR image from the source video into **src**
    b. Create region of interest on **src** as shown below. The pedestrian detection algorithm would run only in this region of interest.

c. The implementation is based on OpenCV's pre-trained HOG + Linear SVM model that is used to perform pedestrian detection in both images and video streams.

d. Initialize the Histogram of Oriented Gradients descriptor by calling **HOGDescriptor hog;**

e. Then, we call the setSVMDetector to set the Support Vector Machine to be pre-trained pedestrian detector by calling the below function
hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());

f. Detecting pedestrians is done by calling to the detectMultiScale method of the hog descriptor.
hog.detectMultiScale(img_roi, found, 0, Size(4,4), Size(0,0), 1.05, 2);
The detectMultiScale method constructs an image pyramid with scale=1.05.

g. This returns set of bounding rectangle coordinates around the detected pedestrians. Filter out overlapping bounding boxes.

h. Use the rectangle() function to draw the blue bounding box.

6. <u>Pedestrian detection warning</u>
   a. A flag **hazard_people** is set whenever a pedestrian is detected in the region of interest shown above.
   b. The flag is used to display the hazard logo along with the type of hazard on the output video.
   c. The hazard logo and the message for pedestrian detection is shown below. The '**P**' indicated pedestrian.

7. Traffic lights recognition
    a. Load BGR image from the source video into **src**
    b. Create region of interest on **src** as shown below

    c. Convert the BGR image to YCrCb image and split the image into separate channels (Y, Cr, Cb).
    d. Traverse the Cr channel to split the RED and GREEN according to the Cr component of these colors. Store the RED in **imgRed** and GREEN in **imgGreen** (imgRed and imgGreen are now binary images)
    e. Dilate and erode to suppress noise in imgRed and imgGreen.
    f. Find contours in imgRed and draw bounding boxes around the contour. If the bounding boxes intersect in consecutive frames, then select this box as the red light. Calculate the area of the resulting bounding box as redCount.
    g. Repeat the same for imgGreen and calculate greenCount.
    h. If redCount > greenCount, then **hazard_light** flag is set which means the lights is RED, otherwise it is cleared (light is GREEN). This flag is used to set the color in the traffic signal window shown in the output video. This flag is also used for the red/yellow light hazard system (which is detailed below).

8. RED light warning
    a. If **hazard_light** is set, then display the hazard logo with the type of hazard on the output video.
    b. The hazard logo and the message for RED light detection is shown below. The '**T**' indicates red traffic light.

9.   Yellow Road sign detection
   a.   Load BGR image from source video into **src**.
   b.   Create an roi **roi_src** as shown below

   c.   Convert roi_src into HSV color space to create **imgHSV.**
   d.   Threshold imgHSV to segment the YELLOW color from the scene.
   e.   erode and dilate to remove noises in the binary image **im_bin.**
   f.   Find the biggest contour in im_bin and fill the contour with white color. Bitwise AND it with the BGR image to produce an image with just the sign.
   g.   Draw a bounding rectangle over the detected sign.

10.   Yellow Road sign warning
   a.   If **hazard_sign** is set, then display the hazard logo with the type of hazard on the output video.
   b.   The hazard logo and the message for YELLOW road sign detection is shown below. The '**S**' indicates road sign.

Output

The following set of outputs are obtained using a prerecorded video data captured from the camera in front.
Sample screenshot of Final output when all the below features are combined

Lane detection
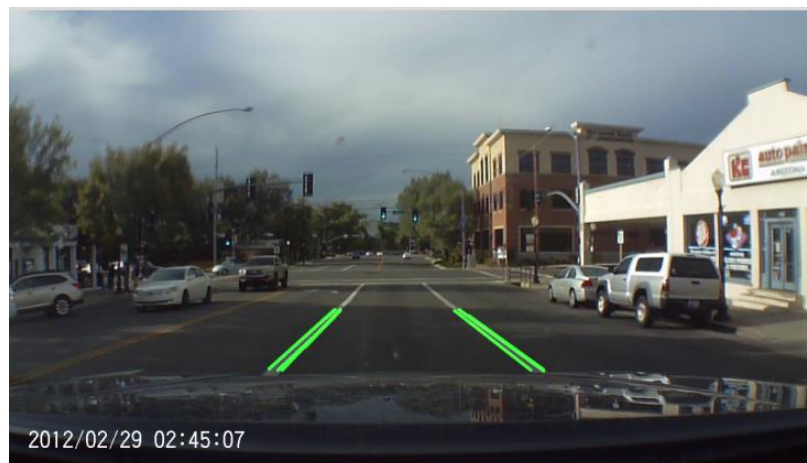
Vehicle detection and distance estimation

Pedestrian detection
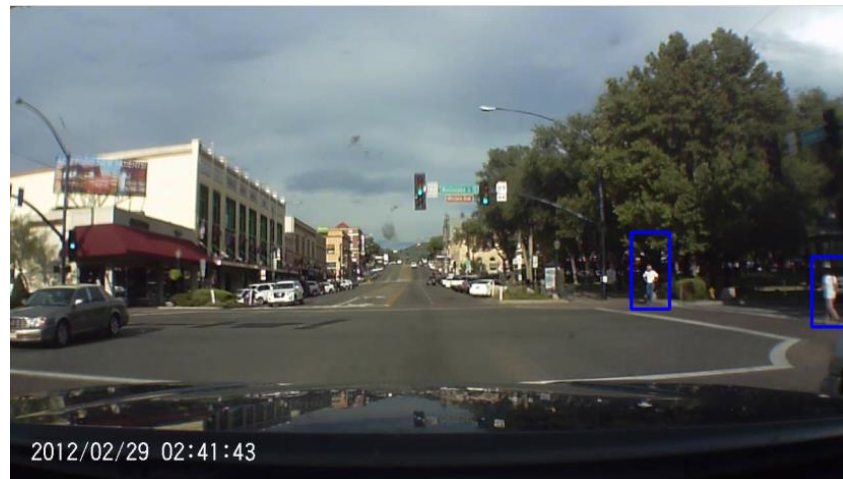
Traffic lights recognition

Yellow road sign detection

Hazard detection
1. Lane crossing

2. Pedestrian detection
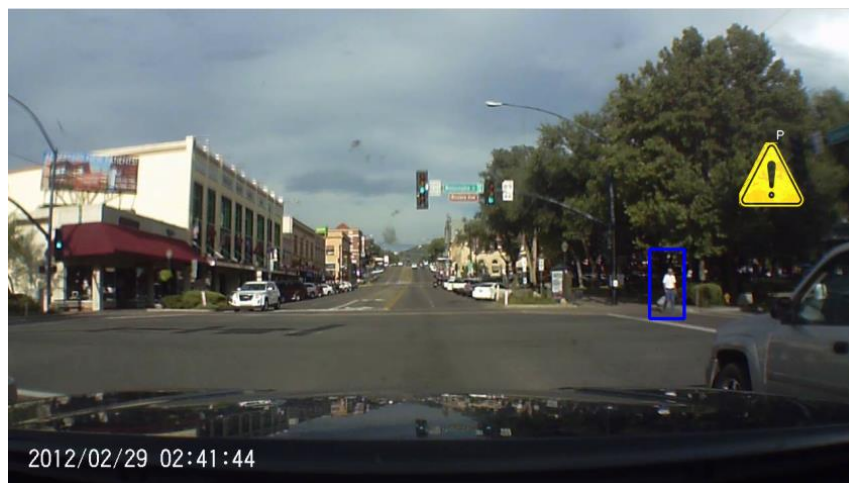
3. RED Traffic lights

4. Vehicle close proximity hazard

5. Yellow Road sign hazard

Performance Evaluation

The performance was evaluated in terms of true positive detections and false positive detections during a time window of 70 seconds.

<u>Lane departure warning</u> (Time stamp info from video)
The false positive detections appear when the vehicle crosses a lane marking on the road such as zebra crossing or other road signs marked on the road. However, all actual lane crossings were successfully detected as shown below.

No:	Actual lane departure instance recorded in video	Result	Detected lane departure instance
1	2:39:30	Detected	perfromance eva\lane_departure\2-39-30.JPG
2	2:39:56 Intersection	Detected	perfromance eva\lane_departure\2-39-56.JPG
3	2:42:03	Detected	perfromance eva\lane_departure\2-42-03.JPG
4	2:42:07 Intersection	Detected	perfromance eva\lane_departure\2-42-07.JPG
5	2:42:38 Intersection	Detected	perfromance eva\lane_departure\2-42-38.JPG
6	2:43:19 Intersection	Detected	perfromance eva\lane_departure\2-43-19.JPG

True Positive vs False positive

T = 70 frames		Actual	
		True	False
Detected	True	44 %	21 %

Vehicle detection (Time stamp info from video) *See constraints for Vehicle detection*
The false detections occur mainly due to the training set which was not well defined. The training set only contained images of cars from the front and rear at a certain angle. This results in numerous false positive detections and failed detection all of which could not be shown here.

Failed detection:

The below table shows some of the true positive detections.

No:	Actual vehicle detected instance recorded in video	Result	Detected vehicle instance
1	2:36:52	Detected	perfromance eva\vehicle\2-36-52.JPG
2	2:36:57	Detected	perfromance eva\vehicle\2-36-57.JPG
3	2:37:08	Partial detection	perfromance eva\vehicle\2-37-08.JPG
4	2:37:20	Detected	perfromance eva\vehicle\2-37-20.JPG
5	2:37:26	Detected	perfromance eva\vehicle\2-37-26.JPG
6	2:37:36	Partial detection	perfromance eva\vehicle\2-37-36.JPG
7	2:37:39	Partial detection	perfromance eva\vehicle\2-37-39.JPG
8	2:38:34	Partial detection	perfromance eva\vehicle\2-38-34.JPG
9	2:38:36	Detected	perfromance eva\vehicle\2-38-36.JPG
10	2:40:02	Detected	perfromance eva\vehicle\2-40-02.JPG

True Positive vs False positive

T = 100 frames		Actual	
		True	False
Detected	True	78 %	3 %

Pedestrian detection (Time stamp info from video)

The below table only shows the true positive detections.

No:	Actual pedestrian detected instance recorded in video	Result	Detected pedestrian instance
1	2:38:17	Detected	perfromance eva\pedestrian\2-38-17.JPG
2	2:38:47	Detected	perfromance eva\pedestrian\2-38-47.JPG
3	2:39:33	Detected	perfromance eva\pedestrian\2-39-33.JPG
4	2:40:51	Partial detection	perfromance eva\pedestrian\2-40-51.JPG
5	2:41:43	Detected	perfromance eva\pedestrian\2-41-43.jpg
7	2:43:17	Detected	perfromance eva\pedestrian\2-43-17.jpg
9	2:44:28	Detected	perfromance eva\pedestrian\2-44-28.jpg

True Positive vs False positive

T = 150 frames		Actual	
		True	False
Detected	True	90 %	10 %

<u>Traffic lights recognition</u> (Time stamp info from video)
The false positive detections happen mainly due to the red tail light of the vehicle in front or when there is a red billboard as shown below

False positive on red billboard

False positive on brake light of the vehicle in front

The below table shows the true positive detections along with one failed detection (No: 15)

No:	Actual traffic light instance recorded in video	Result	Recognized traffic light instance
1	2:36:50 GREEN	Detected	perfromance eva\lights\2-36-50.JPG
2	2:37:10 RED	Detected	perfromance eva\lights\2-37-10.JPG
3	2:37:52 GREEN	Detected	perfromance eva\lights\2-37-52.JPG
4	2:38:38 RED	Detected	perfromance eva\lights\2-38-38.JPG
5	2:39:24 GREEN	Detected	perfromance eva\lights\2-39-24.JPG

6	2:39:52 GREEN	Detected	perfromance eva\lights\2-39-52.JPG
7	2:40:24 GREEN	Detected	perfromance eva\lights\2-40-24.JPG
8	2:40:39 RED	Detected	perfromance eva\lights\2-40-39.JPG
9	2:40:57 GREEN	Detected	perfromance eva\lights\2-40-57.JPG
10	2:41:17 RED	Detected	perfromance eva\lights\2-41-17.JPG
11	2:41:41 GREEN	Detected	perfromance eva\lights\2-41-41.JPG
12	2:42:04 GREEN	Detected	perfromance eva\lights\2-42-04.JPG
13	2:42:37 GREEN	Detected	perfromance eva\lights\2-42-37.JPG
14	2:42:58 RED	Detected	perfromance eva\lights\2-42-58.JPG
15	2:43:14 GREEN	Not Detected	perfromance eva\lights\2-43-14 -false.JPG
16	2:43:53 RED	Detected	perfromance eva\lights\2-43-53.JPG
17	2:44:39 GREEN	Detected	perfromance eva\lights\2-44-39.JPG
18	2:45:09 GREEN	Detected	perfromance eva\lights\2-45-09.JPG
19	2:45:21 RED	Detected	perfromance eva\lights\2-45-21.JPG

True Positive vs False positive

T = 100 frames		**Actual**	
		True	False
Detected	True	72 %	0 %

<u>Yellow road sign detection</u> (Time stamp info from video)
The below table shows all the true positive detections.

No:	Actual Yellow road sign recorded in video	Result	Detected Yellow road sign instance
1	2:36:47	Detected	perfromance eva\signs\2-36-47.JPG
2	2:36:57	Detected	perfromance eva\signs\2-36-57.JPG
3	2:37:08	Detected	perfromance eva\signs\2-37-08.JPG
4	2:37:31	Detected	perfromance eva\signs\2-37-31.JPG
5	2:37:47	Detected	perfromance eva\signs\2-37-47.JPG
7	2:38:05	Detected	perfromance eva\signs\2-38-05.JPG
9	2:38:08	Detected	perfromance eva\signs\2-38-08.JPG
10	2:38:10	Detected	perfromance eva\signs\2-38-10.JPG
11	2:38:36	Detected	perfromance eva\signs\2-38-36.JPG
12	2:39:06	Detected	perfromance eva\signs\2-39-06.JPG
13	2:39:13	Detected	perfromance eva\signs\2-39-13.JPG
14	2:39:23	Detected	perfromance eva\signs\2-39-23.JPG
15	2:39:30	Detected	perfromance eva\signs\2-39-30.JPG
16	2:39:33	Detected	perfromance eva\signs\2-39-33.JPG
17	2:40:00	Detected	perfromance eva\signs\2-40-00.JPG
18	2:42:17	Detected	perfromance eva\signs\2-42-17.JPG
19	2:43:23	Detected	perfromance eva\signs\2-43-23.JPG
20	2:43:35	Detected	perfromance eva\signs\2-43-34.JPG

True Positive vs False positive

T = 180 frames		Actual	
		True	False
Detected	True	27 %	45 %

Since the detection is solely based on color, false positive detection occurs when the roi encounters other yellow objects (shown below)

False positive #1

False positive #2

Future Improvements

This implementation of a self-driving auto-pilot only uses a single camera which captures the visible front view from the car. A future improvement of this project can include using multiple cameras for depth measurement which can be used for accurate estimation of the distance from the vehicle in front.

Since the camera used for this project was a visible range camera, the performance of the project cannot be evaluated during night time without external illumination. Another option is to use infrared cameras so that enough data is extracted from scenes at night.

The project implementation was based on prerecorded video and not real time. In order to make this project meet hard real time system requirements, the program can be parallelized. This would provide enormous savings in execution times since the processes can run parallel and do not have to wait for any previous process to complete.

Acknowledgement

We would like to thank Prof. Samuel Siewert for his valuable inputs and guidance throughout the course without which we would not be able to overcome challenges faced in implementing the project.

Reference

1. Robust Range Estimation with a Monocular Camera for Vision-Based Forward Collision Warning System by Ki-Yeong Park and Sun-Young Hwang
   https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3914606/

2. Train Cascade for Car detection – blog by Abhishek Kumar Annamraju
   https://abhishek4273.com/2014/03/16/traincascade-and-car-detection-using-opencv/

3. HOG person detector tutorial
   http://mccormickml.com/2013/05/09/hog-person-detector-tutorial/

4. OpenCV tutorial
   http://docs.opencv.org/master/d9/df8/tutorial_root.html

Appendix

Source code is attached with this document in the directory **Source_Code.** The input video is not attached with this submission. The link for the video used in this project is given below

http://ecee.colorado.edu/~siewerts/extra/CV-audio/Self-Driving-Car-Video-2-Raw/Visible-front-facing/02260002.AVI
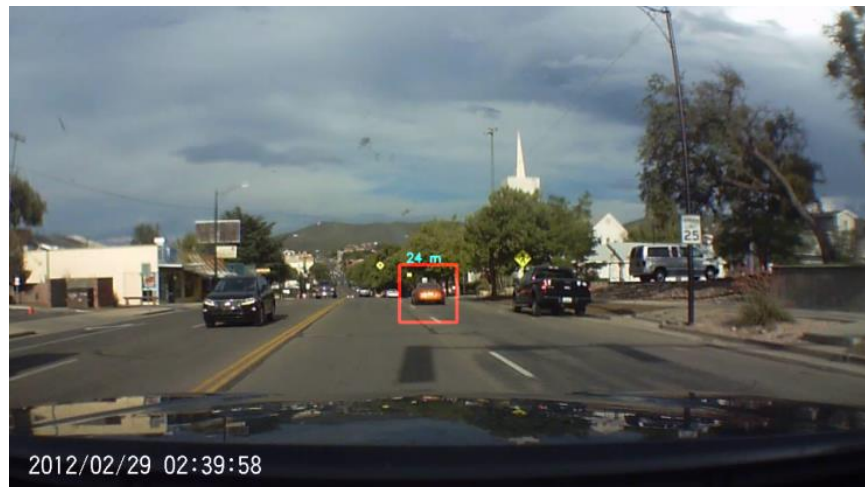
Additional output screenshots

Lane detection #1

Lane detection #2
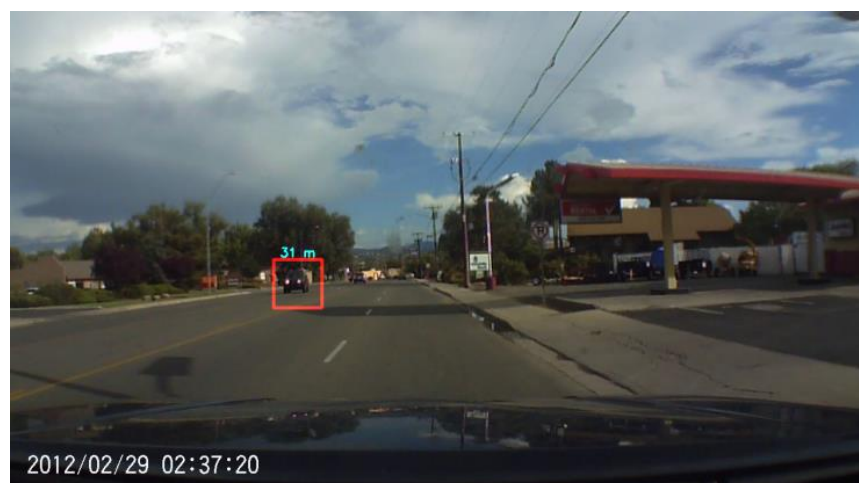
Lane detection #3 - Rejecting horizontal lines among road lanes

Vehicle detection and distance estimation #1

Vehicle detection and distance estimation #2

Pedestrian detection #1

Pedestrian detection #2

Pedestrian detection #3

Traffic lights recognition #1

Traffic lights recognition #2

Traffic lights recognition #3

RED light hazard #1

RED light hazard #2 - False positive from rear light of the car in front
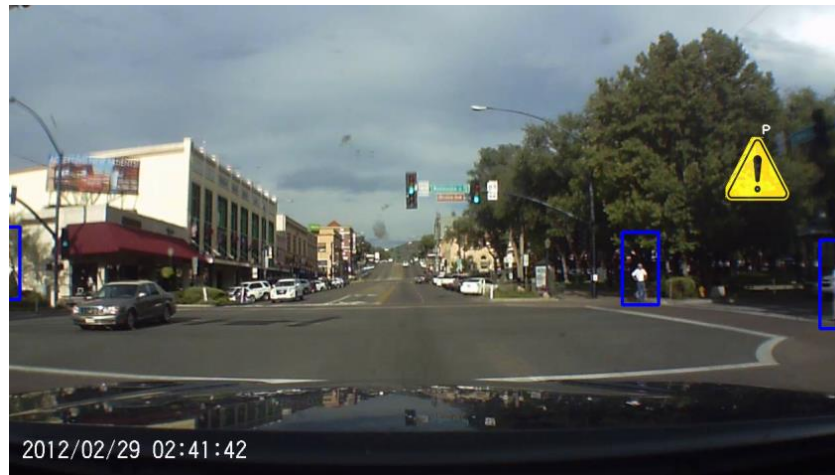

Lane departure hazard #1

Pedestrian hazard #1

Pedestrian hazard #2

Yellow sign detection #1

Yellow sign detection #2