Mini Project in Robotics
on

Autonomous Car Project - Vision Implementation

Submitted by:

Harman Mehta - 2016BB50003
Tanmay Goyal - 2016ME20757

Supervisor:
Prof. Sunil Jha

Department of Mechanical Engineering

Indian Institute of Technology, Delhi

New Delhi - 110016

Acknowledgement

We would like to express our deep gratitude to Professor Sunil Jha, our project supervisor for his patient guidance, continuous encouragement and useful critiques for this project. We also thank Sidharth Talia for cooperating with us at so many points and helping us out during the project. We also thank Mahindra for providing us the technological resources to pursue the project.

Abstract

In recent years, a lot of research has been conducted on Autonomous cars and control. This report discusses a section of the parent project, based on the development of an Autonomous Controller for the self-driving Mahindra-e2o. It talks about implementation of vision, perception and navigation for realizing autonomous driving. The primary aim is to design a lane keep assist system which has two components, namely, perception (lane detection) and motion planning (steering). We turn a video of the road into the coordinates of the detected lane lines and generate a continuous optimal way point for the car to achieve driverless control.

We have discussed our approach for lane following and driving, the theory and literature behind it, simulations on ROS Gazebo for testing the technology, qualitative comparison of the designed models, live ZED stereo camera feed testings of the algorithms and finally generating relevant insights and conclusions. Getting encouraging results from implementing the vision-based lane detection algorithm on a live video stream, we integrated the algorithm with car steering control, thus enabling the car to move autonomously on a straight road from vision feedback for the first time ever.

Contents

List of Figures

Chapter 1: Introduction

1.1 Background

The automobile industry seems to be on the brink of a giant technological leap. It has come a long way since the development of the first commercial car in the early 20th century. Autonomous / Self driving / Robotic cars have recently attracted a lot of attention all round the world. An autonomous car can sense its environment, interpreting different sensory information (LIDAR, GPS, Computer Vision) and guide itself safely with little or no human input.

The world's first radio-controlled car was 'Linrrican Wonder' (1926). Over the years many companies like Mercedes, Google, Nvidia, Tesla have invested heavily in autonomous cars and significant progress has been made. There are broadly 5 different levels of autonomous driving -

Level 0 : The system can only issue some warnings based on certain sensor readings. The control remains mainly manual
Level 1 : The automated system can control certain systems like steering system during Parking Assistance ; speed during Cruise Control
Level 2: Partial automation. The system can control the steering, speed, breaking of the vehicle but the driver has to continuously monitor the car
Level 3: The driver can take his/her eyes off, but the driver must be ready to take control within a specified time interval
Level 4: No human attention is needed but the driver still remains in the car. The automated system should have the capability to follow all safety protocols.
Level 5: 100% automation. No driver in the car.

Some of the recent autonomous systems launched could operate at Level 3 and 4. The driverless system we are implementing works at the Level 2, where the car automates all the control systems but under continuous monitoring of the driver. In this report, we implement our autonomous system using softwares like OpenCV, ROS and Gazebo which could simulate real life data using sensors and virtual environment.

Autonomous vehicles can help solve major problem faced in India - traffic jams, accidents (93% of accidents are due to driver negligence), environment impacts, parking space etc but the unique nature of chaotic, diverse Indian roads and transport pose a big challenge in designing successful self-driving cars.

1.2 Overview

In the thesis, we have designed a lane keep assist system, initially understanding and working on raw color and image detection edge detection algorithms for images. Later we tested them on ROS (Robot Operating System). A simulation environment with a turtle bot along with its proper control inputs had already been created for the project. We used the 3D physics engine Gazebo to create this environment. The reason behind using Gazebo is due to its high flexibility and its easy integrating nature with ROS.

Later we improved our lane detections algorithms by employing various strategies like feature detection and filters. We tested these on data from Zed Stereo camera: mounted on the windshield pointing towards the front of the car. We tested it on all the roads inside IIT Delhi. Following this we planned the lane following motion of the car on straight, curved and discontinuous laned roads. Currently we are getting good lane detection results. We are now focusing more on improving the control of the car and the correction algorithm used.

Chapter 2: Literature survey

2.1 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. The library has over 3000 optimized computer vision and machine learning algorithms which provide space for computer vision applications and commercial machine perception models. These algorithms can be used to detect faces, identify objects, recognize and classify human actions, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras.

2.2 ROS

ROS is an open-source, meta-operating system for robots. It provides services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. It provides tools and libraries for building, writing, and running code across multiple computers.

ROS-based processes work on a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator, and other messages.

2.3 Gazebo

Gazebo is a 3D multi-robot simulator, complete with dynamic and kinematic physics, and a pluggable physics engine. Set of Gazebo plugins provide integration between ROS and Gazebo that support many existing robots and sensors. Since the plugins present the same message interface as the rest of the ROS ecosystem, you can write ROS nodes that are compatible with simulation, logged data, and hardware. You can develop your application in simulation and then deploy to the physical robot with little or no changes in your code.

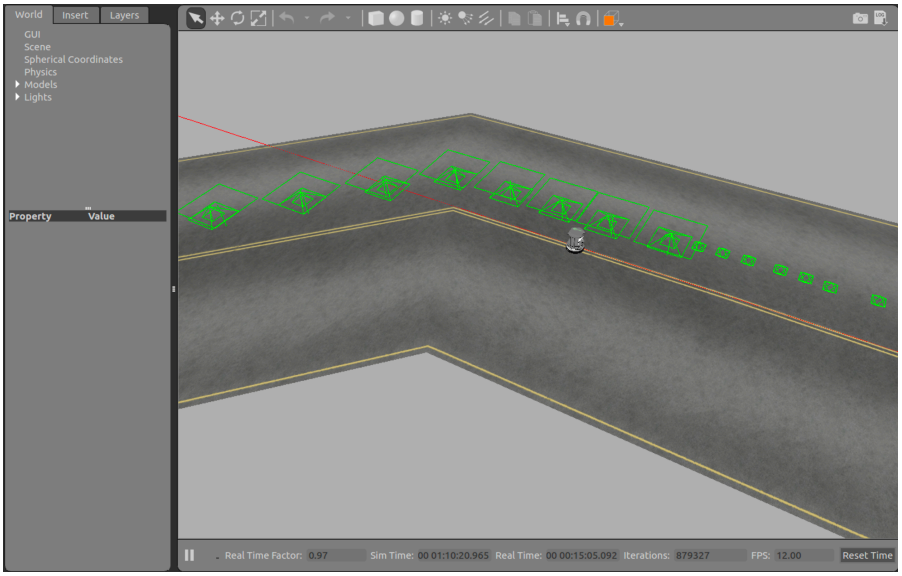


Fig 2.1 Gazebo simulation platform

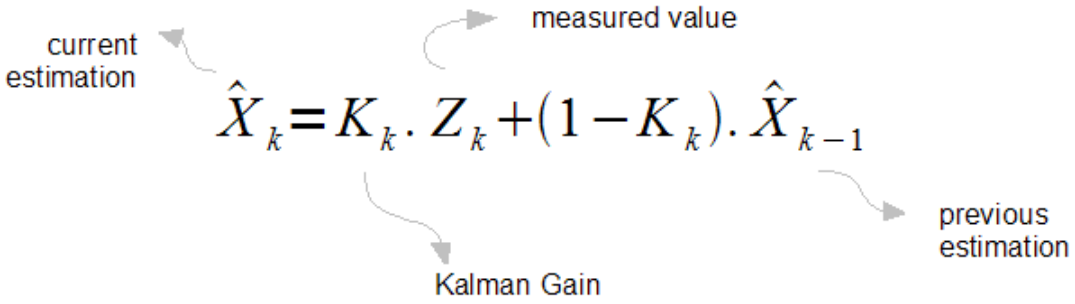
2.4 ZED Stereo Camera and Nvidia CUDA

ZED Camera is a product of Stereo Labs. It is a 3D camera for depth sensing, motion tracking and real-time 3D mapping. It provides wide-angle all-glass dual lens image with reduced distortion.To process ZED feed (SVO format) to AVI format, SDK requirements need Nvidia GPU with compute capability > 3.0, for this we used Nvidia CUDA (Compute Unified Device Architecture). The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

2.5 Kalman Filter

The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown. The algorithm works in a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed,

these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. The algorithm is recursive. It can run in real time, using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required



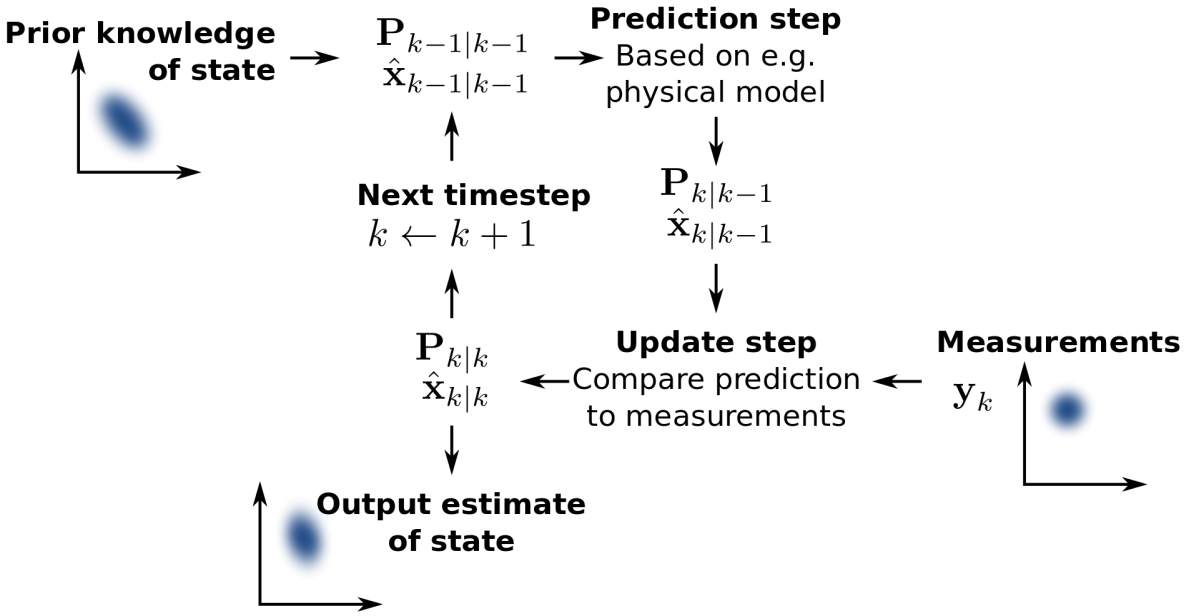


Fig 2.2 Kalman filter working

Chapter 3: Project Objectives and Workplan

3.1 Problem Definition/ Motivation

Recently, Mahindra took an initiative to encourage research on autonomous vehicles in India through the Mahindra Driverless Car Challenge. The challenge is divided into levels. Level 0 challenge includes simple tasks such as lane driving and reading traffic signals. The complexity of the challenge increases and level 3 will address problems such as avoiding a truck with protruding rods, identifying shallow ditches and unmarked bumps, wading through waterlogged streets and night driving capabilities. Team dLive at IIT Delhi is participating in this competition and is currently working together to develop a fully autonomous vehicle.

3.2 Objectives of the work

Our objective is to move a driverless car on road at Level 2 stage of autonomous driving with the help of lane following.

Goals of the project are:

- Implementing the optimal perception (lane detection) model on the video feed
- Testing on the real-world live feed on Mahindra e2o
- Design the motion planning (steering) for lane following
- Improve the control and correction algorithm of the car motion

A GitHub repository is also being maintained for the project so that we can maintain multiple versions of the same project and make it easy for others to contribute as well.

Link - https://github.com/tanmay2798/Autonomous_car.git

3.3 Methodology

The goal we opted for is to make the car move autonomously using vision system feedback. First, we started studying the existing methods that were available online. We after surveying numerous methods and extensive discussion concluded that autonomous navigation can be done using the lane mark detection and prediction technique using OpenCV Kalman filter. So, for the first month we worked on the problem of solving lane driving in a simulated environment - Gazebo. The next task at hand was to implement the PID control system on the Turtlebot in simulated environment taking input from lane detection to verify the algorithm.

After successfully implementing the control in simulation, the next task in hand was to implement the lane detection system in the real world taking care of all the noises, disturbances and interfering objects. This will help us in providing a robust input of detected lanes and direction to steer towards the autonomous car in real-world hence achieving autonomous movement.

Below is the Gannt chart representing some of the objectives we have completely achieved and also our future work plan.

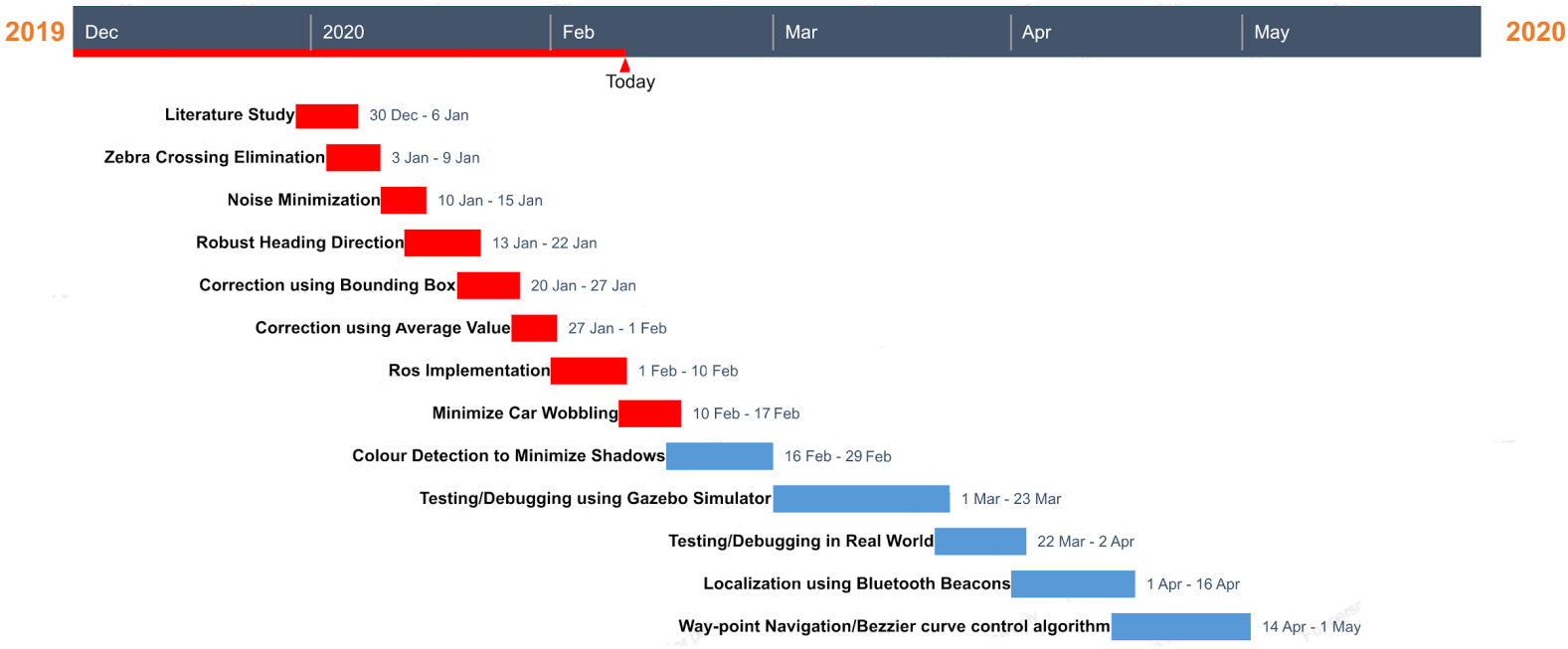


Fig 3.1 Gantt Chart

Chapter 4: Previous Work

4.1 Region of Interest

Even after applying Canny Edge Detection and Hough Line Transform, there are still many lines that are detected which are not lanes. To solve this issue, we selected the Region of Interest from the frame to exclude all the lines which are not lanes.

Region of Interest is a polygon that defines the area in the image, from where edges we are interested. The coordinate origin in the image is the top-left corner of the image. Row coordinates increase top-down, and column coordinates increase left-right.

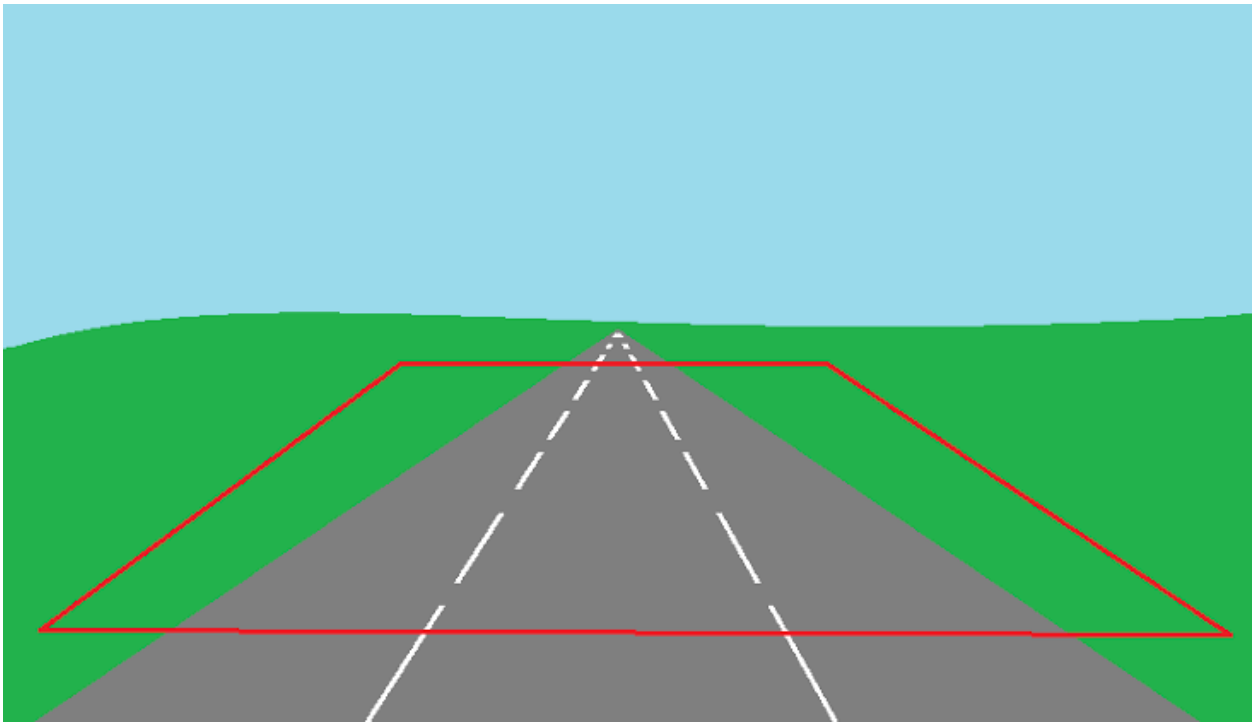


Fig 4.1 Region of interest inside the polygon

Testing 1

1) We cropped out the region of interest excluding all the surrounding items like trees, sidewalks, buildings, humans etc. which provided interference to the detection.

2) We then took a mobile camera to record a video and implemented the algorithm on the video getting promising results with slight interference of the surroundings.

3) The results were accurate but the code loop time was too long because of the each pixel of the image being scanned out to find the lane. Thus implementing this on a real car can lead to very slow refresh rate and response.

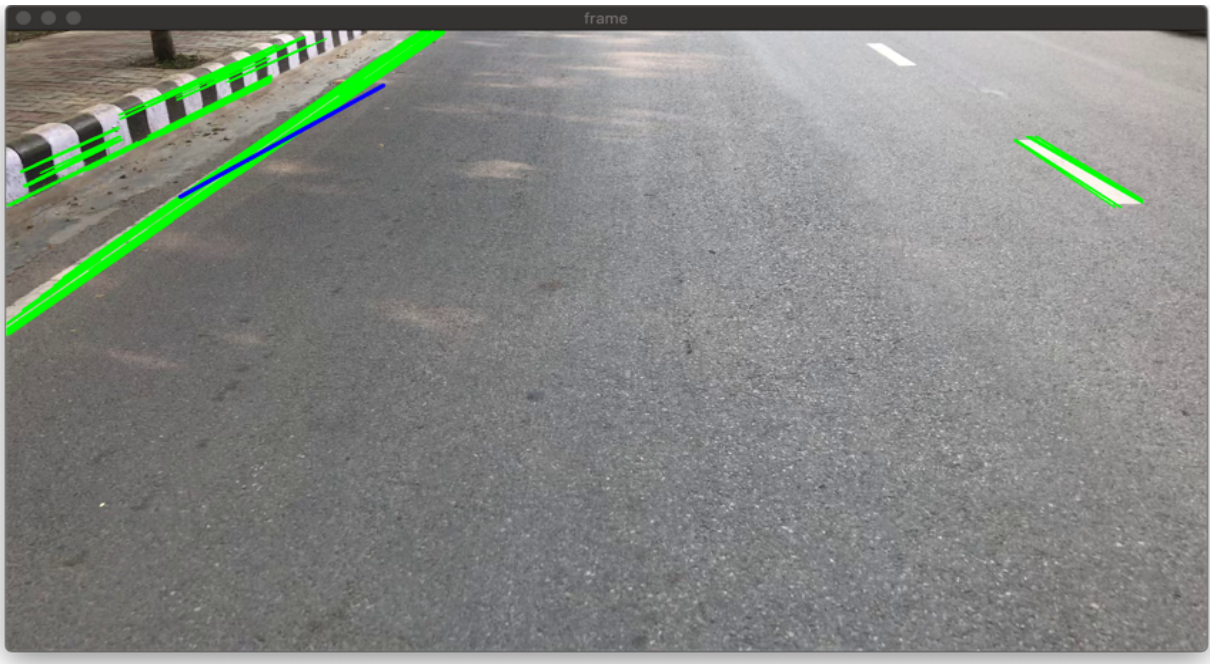


Fig 4.2 Lane detection on video recorder from mobile

4.2 Perspective Transform

We now need to define a trapezoidal region in the 2D image that will go through perspective transform to convert into a bird's eye view, like below:

Fig 4.3 Trapezoid made using the detected lanes

We then define 4 extra points which form a rectangle that will map to the pixels in our source trapezoid:

```
np.float32([[569, IMAGE_H], [711, IMAGE_H], [0, 0], [IMAGE_W, 0]])
```

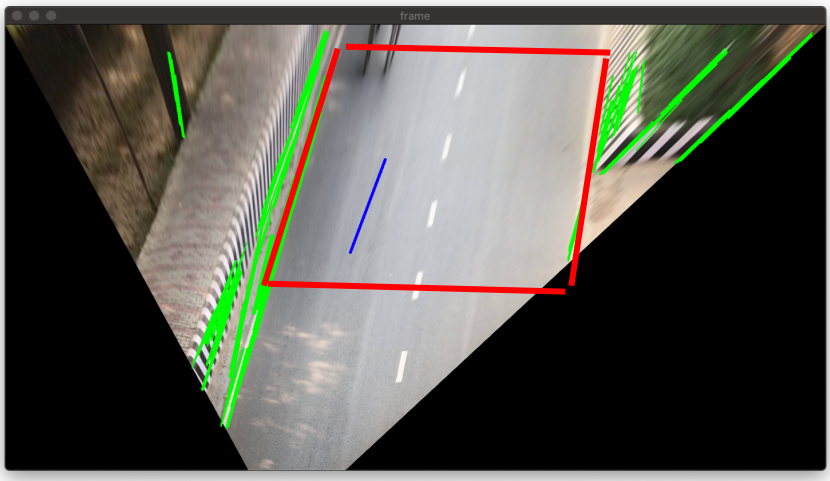


Fig 4.4 Rectangle in bird's eye view

4.3 Revised Lane Detection Algorithm

We incorporated a few changes to the lane detection algorithm in order to overcome it's limitation of slow loop speed and inaccurate lane detection because of the noise and interference of objects. The changes made were -

1) **HoughLines instead of HoughLinesP** - This increased the loop speed significantly by reducing line detection time.

2) **Detection of left out lines** - We started detecting those lines as well, of which only one end and slope can be detected in the image frame and other end lies somewhere outside the image.

3) **Scaling Lines** - We scaled the farthest point of the line segment to be on the drawing horizon so that the point is not missed out. In this way we were able to detect all the lines that can be found in a frame for getting better results.

By this change we were also able to detect very accurately the curved roads and discontinuous lane marks, thus solving the issue of turning and improper lane marks as well. We also solved the issue of shadows and lighting conditions by this and made a system robust to lighting conditions

Fig 4.5(a) Discontinuous lanes detected

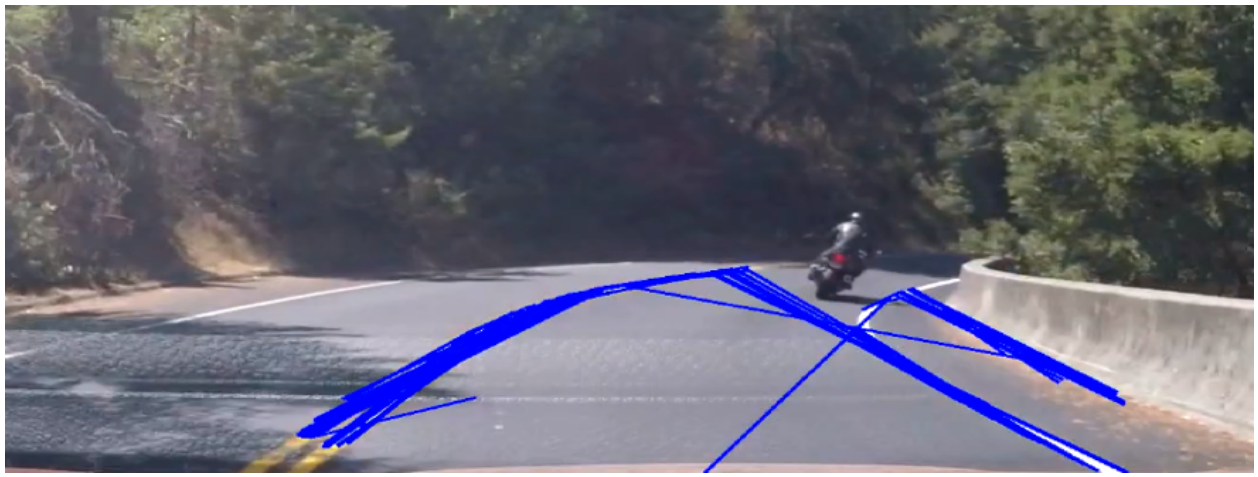


Fig4.5(b) Lane detection on curves

Testing 2

1) Due to some network/communication error, most of the ROS packaged dropped, and hence we had very less live ZED data.

2) We got only 3 frames and we tested our new algorithm on those 3 frames getting more lines detected than earlier even in dim light conditions.

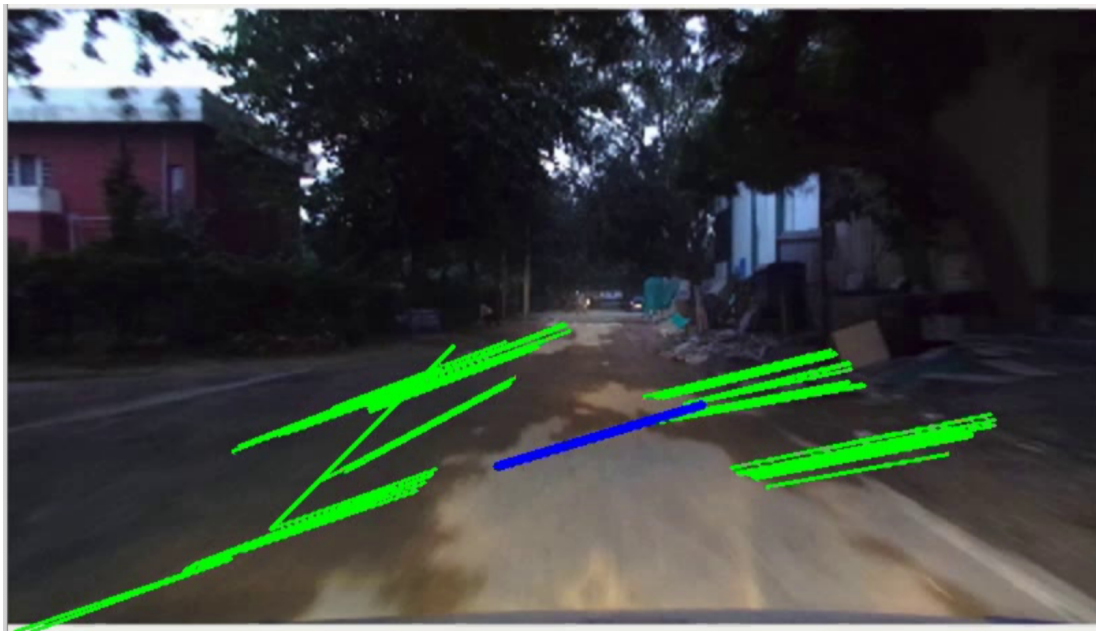


Fig 4.6 Edge detection by new algorithm

Testing 3

1) We conducted a test run using ZED camera, recorded the data in SVO format using ZED Explorer software. The SVO file format is a proprietary format that can only be read from the ZED SDK and its tools. It contains the unrectified images of the camera along with metadata information such as timestamps and IMU data. To convert it to AVI or MP4 format on which lane detection could be carried out, we used the Nvidia CUDA 10.0.
2) We received promising results on running the algorithm on the recorded avi videos

Fig 4.7 Lane detection on ZED video feed

4.4 Kalman Filter Approach

After detection of lanes accurately in lighting conditions, we faced some major issues to be solved for making the system more robust. The issues were -

1) Low light intensity in some parts of the day and at night (Due to absence of streetlight).
2) Difficult to tune the parameters for various light intensities.
3) Poor edge detection(irregular lane availability or absence of road lanes)
4) Shadows, Sudden high intensity from other vehicle headlights.

Thus we shifted our approach to a more sophisticated one in which we started predicting the lanes from current data and error data. This solved the issue of discontinuous lane marks completely and also to some extent the issue of turns where lane marks are not present. We used Kalman Filtering for better lane detection. Here, we applied as a linear estimator for the clustered lines for the lanes to make it stable and free from any offset errors.

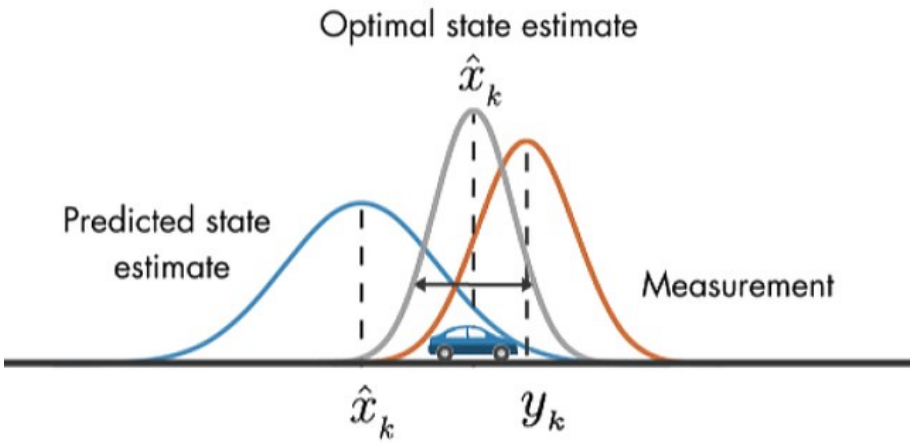


Fig 4.8 Optimal state estimation by Kalman

1) The Kalman filter is an algorithm that uses noisy observations of a system over time to estimate the parameters of the system (some of which are unobservable) and predict future observations. At each time step, it makes a prediction, takes in a measurement, and updates itself based on how the prediction and measurement compare.

2) The LaneTracker class implements the Kalman filter for lane detection. Firstly, it initializes the State matrix & Measurement matrix size. Then, it calculates transition matrix. We take White Gaussian noise for our system. Using this noise model, we have calculated the error for state and used in estimator to generate the predicted state using that measurement noise.

3) Variation of detected lines along the lanes are averaged out by the Kalman filter by adding up the measurement error and previous state. That's why detected lane marker lines are stable over time and for its predictor property from previous state, at very low illumination condition, it can be able to detect lanes by remembering previous detected lanes from the previous video frame.

Testing 4

1) We recorded the new HD videos in better lighting conditions by ZED camera, converted them to avi and tested Kalman filter on it.

2) We specially recorded those parts of IIT Delhi where lane marks were not present, or curves were there to test robustness.

3) We got significantly greater results with **absolutely 0** region of no predicted lines. Thus, the car will have a direction even if no lane marks present to some extent until Kalman filter gets affected by extreme noise.

4) The image shown below justifies that even at turns of no lane marks, the red predicted lines gives the direction of curve to be followed by the car.

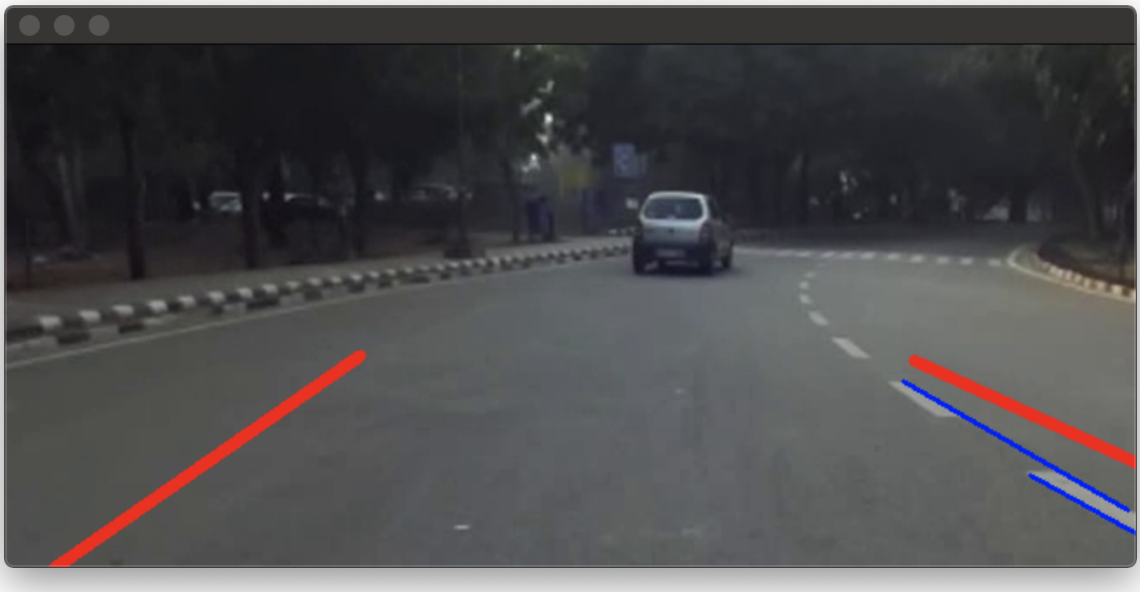


Fig 4.9 Karman filter on curves

Chapter 5: Lane Detection in Real world

5.1 Zebra Crossing exclusion

1) We eliminated the zebra crossings detected to avoid its interference with lane detection.

2) This was done by ignoring lines that were horizontal, i.e., had an absolute angle less than 5 degrees with horizon.

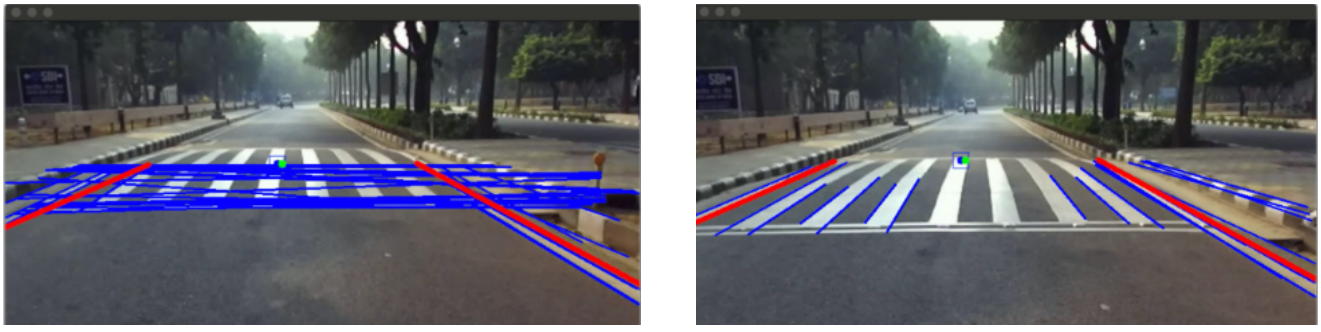
3) We reduced the noise observed in lane detection by ignoring lines which had an absolute angle difference greater than 20 degrees with the line predicted by the Kalman filter for both left and right boundaries of the lane detected.

Testing 5

1) We recorded the new HD videos in better lighting conditions by ZED camera, converted them to avi and tested Kalman filter on it.

2) We recorded parts of IIT where zebra crossings were there to test robustness.

3) We found that zebra crossings have been successfully excluded and thus the horizontal lines didn't deviate the predicted kalman filter lines.



A **B**

Fig 5.1: Zebra crossing detection
A) before elimination B) after elimination

5.2 White colour detection

1) We were getting issues of detecting lines in shadows, footpaths and other obstacles that were interfering with our algorithm and giving incorrect values. To solve this, we used a colour detection algorithm in which we detected a range of white colours in the video captured and then processed only those areas for lanes.

2) This gave us better results as footpaths and other areas were ignored. But this posed a few major issues -

    a) As we are traversing the entire image pixel by pixel to find white region, this reduced the speed of the algorithm significantly increasing the time of giving output. As a result, car movement couldn't be handled.

    b) Also, the colour density of side lanes was dependent significantly on sunlight and also varied highly for different side lanes. Therefore, the colour code tuned for a particular side lane will be of no use for other side lanes.

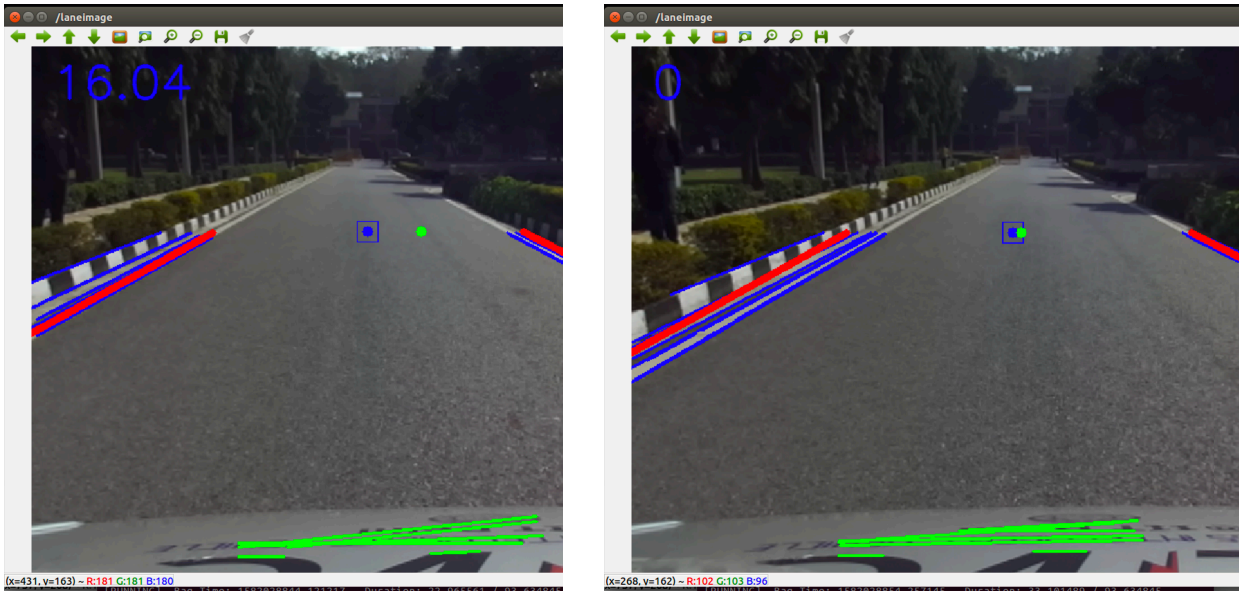
3) As a result, this approach was discarded because it was slow and dependent on sunlight and side lanes highly.

Testing 6

1) We used the recorded HD videos of the road in front of the security room and tuned the colour code for it. But this colour code failed for the previous video recorded behind the blocks and slowed the code.

2) In the image shown below also, in Fig 5.2 A, we detected white colour of the left side of lane and the RGB values were (181,181,180). On the other white colour of the right side of lane had RGB values (102,103,96) as shown in the Fig 5.2 B.

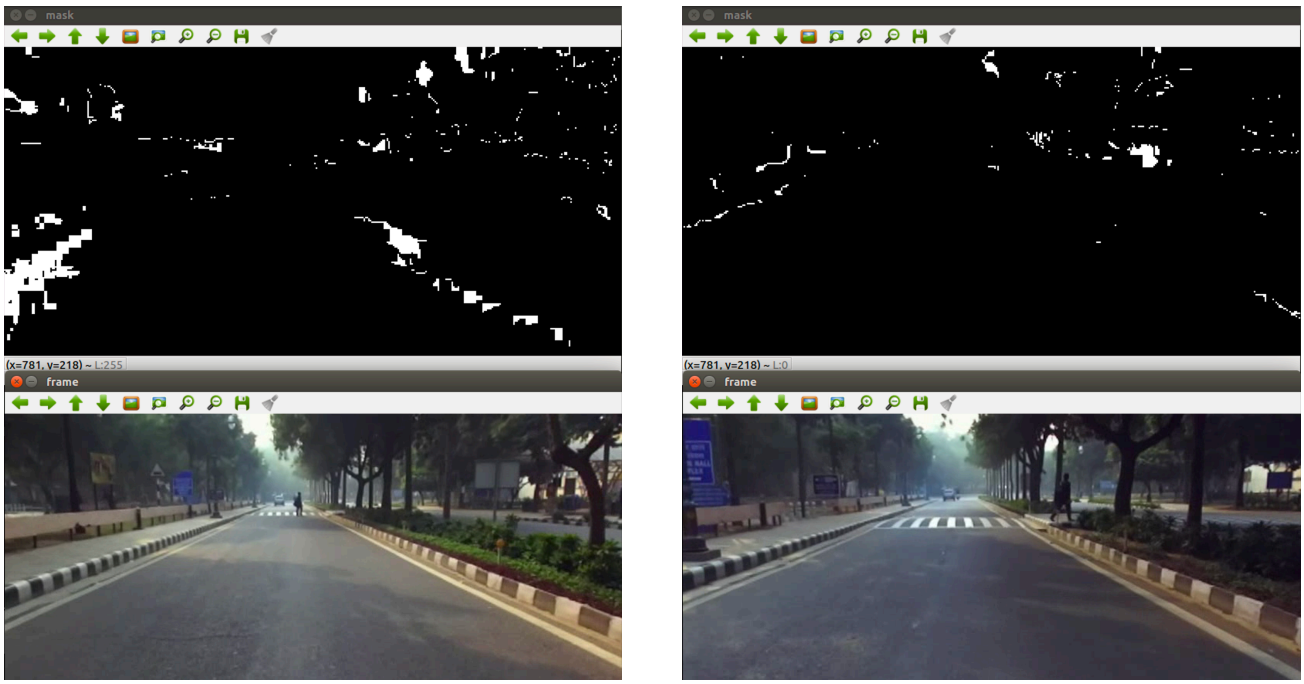
3) In the image shown below also, in Fig 5.3 A, we detect side lines well because of white colour detection, but were not able to detect the same 100 meters away on same road as shown in Fig 5.3 B.



A **B**

Fig 5.2: Variation in white colour

A) Left side of lane B) Right side of lane



A **B**

Fig 5.3: White colour detection

A) Initial part of path B) 100 meters away on same road

Chapter 6: Control in Real world

6.1 Motion Planning: Steering with instantaneous values

Knowing the exact location of lanes, we needed to steer the car so that it stays in the middle of the road. We designed an algorithm to give a steering angle to the car from the detected lanes.

In the figures below -

Green pointer - X coordinate = (ZED frame width) / 2 - (6.1)
- Y coordinate = (avg Y coordinate of lane lines) - (6.2)

Blue pointer - X coordinate = Avg[min(X coord of left lane) , max(X coord of right lane)] - (6.3)
- Y coordinate = (avg Y coordinate of lane lines) - (6.4)

The green pointer represents the heading direction (the dashcam or ZED camera is in the middle of the car) and blue is the desired direction of motion. Therefore, we need to move in the direction that green follows bue pointer. Ideally, they both should coincide.

$$\tan(\theta) = (x(\text{blue}) - x(\text{green})) / (\text{avg Y coordinate of lane lines}) \quad - (6.5)$$

A negative θ means a left turn, and a positive value means a right turn.
Once this θ is achieved, we map the rotation angle of steering wheel with it's value to get the angle that the steering wheel should rotate with.

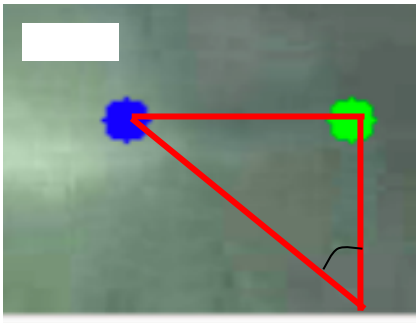


Fig 6.1 Calculation of steering angle

Testing 7

1) Tested our motion planning algorithm by subscribing to rostopic "/zed/stereo/image_rect_color" realtime, generated using the ZED Stereo camera mounted on the car.

2) This gave us the image frame in the form of **img_msg** which was converted to **cv2** form for further processing in OpenCV.

3) The Fig 6.2, shows that the desired direction of motion is towards left, as verified visually and using the equation below we generate the angle of steering θ. A steering angle of -12.4 degrees (to the left) is given as output by the algorithm which is published to car controller.

Fig 6.2 Steering angle generated on curved road

4) On a straight road, both the points nearly overlap, and the calculated angle is 0.51 degrees (Fig 6.3). To solve this, we set a minimum threshold for the car to make a decision, thus it keeps moving straight in case of small errors.

Fig 6.3 Steering angle on straight road

5) Once this angle was found, it was published on a rostopic “/steer” which then can be used later on by the autonomous car control to get the steering rotation angle.

Testing 8

1) Conducted a test of lane following on the road behind IIT Delhi blocks by integrating the steering angle with the control system of the car.

Fig 6.4 Live demonstration

6.2 Robust Heading Direction

Our previous method of finding the heading direction was not robust as it was highly dependent on the width of the road. To solve this issue, we used the front part of the car as a reference for the heading direction. We divided the frame into lower and upper half by frame coordinates -

1) **Upper half** – We used only the upper half of the frame for the lane detection. This allowed us to neglect all the horizontal lines that are encountered from the front of the car. This is then used to predict the direction of motion. (blue lines)

2) **Lower half** – We used the lower half of the frame for detecting the horizontal line on the front of the car to detect the car center. By it we got the heading direction of the car. This heading direction of the car was independent of the width of the road and frame we selected. (green lines)

Testing 9

1) We tested this approach to find the heading direction of the car (green dot) on roads of different width and were able to find it changing dynamically according to road width making system highly robust.

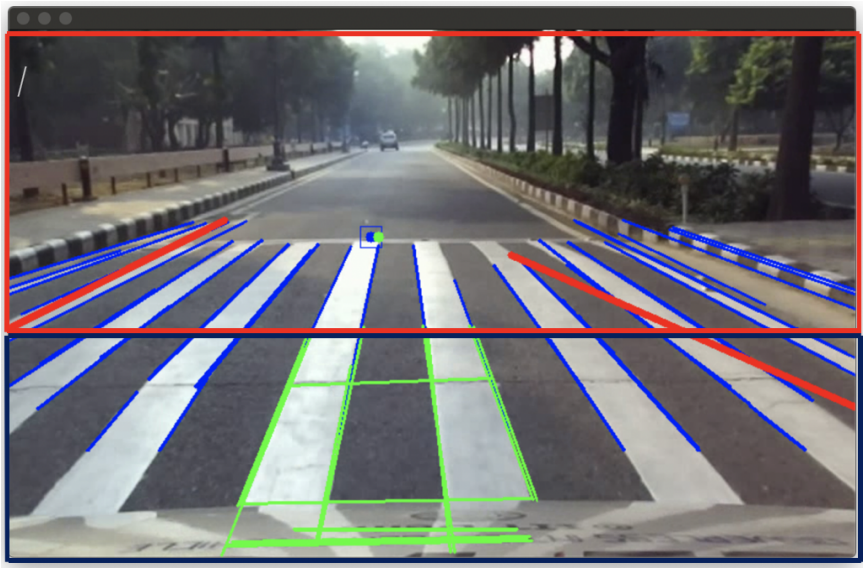


Fig 6.5 Divided Frames approach

2) But the center of the car and the road we calculated from this varied highly because of certain wrongly identified lines giving incorrect values. This made the car wobble a lot.

6.3 Control using One Side of Lane

After careful deliberation and discussions, we decided to try a different approach for controlling the car using lane detection. We followed the approach of one-sided lane detection. We took only the left side of the lane and coded the car to always move at a fixed distance from the left side. In this way we gave corrections to the car so that ideally it can move on an imaginary straight line parallel to lane marks on the left side.

But it posed some issues as it had a lot of variations and incorrect readings many times due to variations in the predicted line. It was also dependent on road width and was a problem when one side had no lane marks. It was also a problem at turns and roundabouts where only one side had lane marks.

Testing 10

1) We tested this concept on the pre-recorded videos. But we observed a lot of variation in the values we were getting to steer.

2) Also, of initial value we get are wrong, then the correction was an issue as there was no right side value to average out and eliminate the errors.

Fig 6.6 One sided lane detection results

6.4 Minimizing Car wobbling using Bounding box

In order to reduce the variations of the centers of the car heading direction and road, we ignored the values of centers that were more than a specified distance from the initial value as such a variation meant incorrect center. But this only reduced the incorrect values but didn't reduce the close variation in centers.

To solve this issue, we created a contour (20 * 20 pixels) around the center of the road and only published a steering angle for the car if the center of the heading direction was outside the contour. This reduced the centers robust to small deviations in values but after a certain value, it varied highly.

Also, in this approach if the center is in some case stored at a side, then its correction was an issue as the initial value was wrong. So, we had no correct basis to measure the distances.

Testing 11

1) We tested this on the recorded videos and observed that this bounding box can reduce the steering wobbling for very small deviations giving a better control of the car.

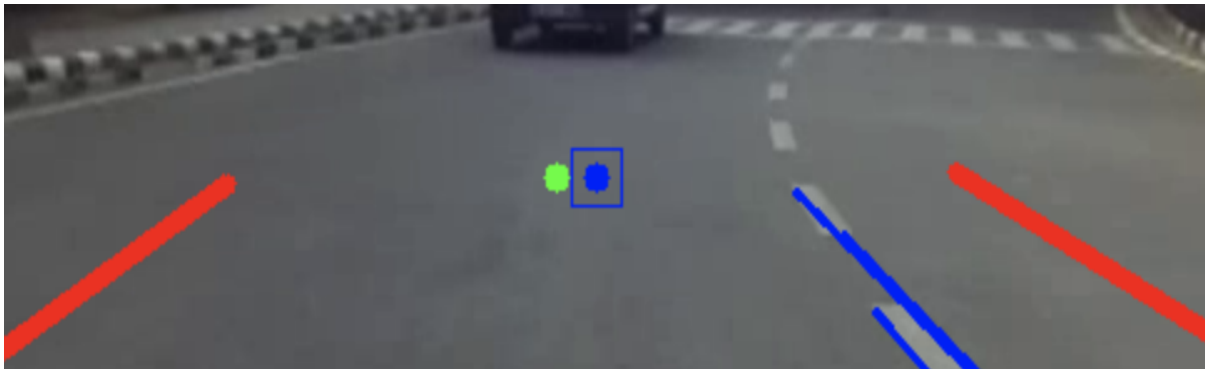


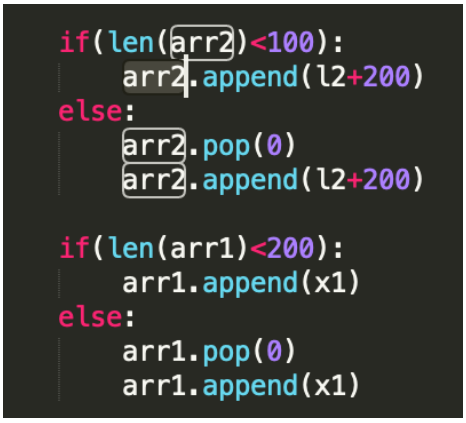
Fig 6.7 Bounding Box

6.5 Minimizing Car wobbling using Average values

In order to reduce sudden variations in center detection, we started storing the x coordinate of the center of the road (blue dot) and the heading direction of the car (green dot) in 2 separate arrays of length 200 and 100 respectively.

When there was no sudden deflection in the detected value, we updated the array by adding the new detected value and removing the oldest value in the array. Then we used the average of all the values in both the arrays for control instead of the instantaneous values.

This posed a problem when the points deviated, and wrong values were inserted in arrays. As we are taking average, it took a while for the points to be corrected. To solve this, if wrong values are inserted in arrays giving a turning angle of more than 10 degrees instantaneously, we flushed all the values of array to start a new set of values, so that incorrect values are not used for averaging.



```
if(len(arr2)<100):
    arr2.append(l2+200)
else:
    arr2.pop(0)
    arr2.append(l2+200)

if(len(arr1)<200):
    arr1.append(x1)
else:
    arr1.pop(0)
    arr1.append(x1)
```

Fig 6.8 Code for using Arrays as Queues

$$x\_avg = sum(arr1)/len(arr1) \quad -(6.6)$$

$$l2\_avg = sum(arr2)/len(arr2) \quad -(6.7)$$

$$dif = l2\_avg - x\_avg \quad -(6.8)$$

Testing 12

1) We tested this concept on the car in front of the security room, IIT Delhi. We were able to move the car which was correcting itself on a straight road for 100 meters.

2) The correction was a problem when any person or shadows came on the way. This was because of the extremely slow speed of the car which gave a lot of time for the errors to get accumulated and give wrong corrections.

3) When we increased the car speed, we were able to cover more distance. But because of the limit of the car speed in the real world, we are planning to test and debug further in the Gazebo simulator.

Fig 6.9 Mahindra e2o car live testing

6.6 Rejecting the Bounding Box Method

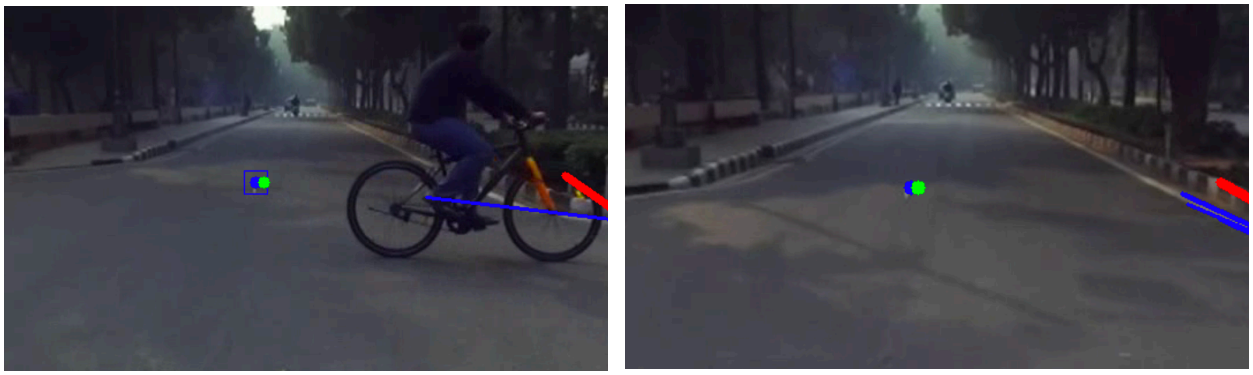
In order to reduce the wobbling nature of correction of heading direction of the car, we earlier adopted the bounding box approach. But we observed that this was leading to the car direction not being corrected completely and the car instead kept moving at a non-zero very small angle with the heading direction. This eventually led to significant deviation in the long run of the car.

Hence, to overcome this error in control in order to prevent the car moving at a non-zero small angle, we removed that bounding box and kept using the average value method for minimization of wobbling.

Testing 13

1) We used the recorded HD videos of the test run to test correction without the bounding box.

2) We observed that although the car kept on getting very small correction angles continuously, we were able to solve the issue of very small angles car made with the heading direction the heading direction coincided perfectly with the path direction.
3) In Fig 6.10 A, the heading direction does not exactly coincide with the center of lane thus having a small angle. But in Fig 6.10 B, both coincide exactly having 0 degree angle with each other.



A **B**

Fig 6.10: Rejection of bounding box
A) With bounding box B) Without bounding box

Chapter 7: Implementation using ROS

7.1 Input from ZED camera

We connected the ZED camera to the system and installed ZED wrapper libraries in the system. We also installed NVIDIA drivers and CUDA 10 which were needed for the ZED camera to run in system

We launched the **zed.launch** file in the system that starts all the rostopics of ZED camera and setups a communication between ZED camera and the system.

roslaunch zed_wrapper zed.launch -(7.1)

We used the rostopic - "**/zed/zed_node/stereo/image_rect_color"**. This rostopic publishes the stereo image captured by ZED camera which can be subscribed to get the input from ZED camera.

7.2 Input to python code

We took this image rostopic as the input to our python code. We initiated a node - **"image_converter"**. Using the node we subscribed to rostopic - "**/zed/zed_node/stereo/image_rect_color"**.

rospy.init_node('image_converter', anonymous=True) -(7.2)

We got the input image as a ros message. To process it using OpenCV, we converted into cv2 type which can then be processed for lane detection.

frame = self.bridge.imgmsg_to_cv2(data,'bgr8') -(7.3)

We cropped the frame according to the need to remove all the disturbances due to objects on the sides of roads.

7.3 Output from python code

Once the image was processed and we were able to find the steering angle using the control methods as discussed in chapter 5, we needed to publish it to the car, so that the steering wheel can rotate through the specified angle.

We started a rostopic - **"/steer"** of Float32 data type. We published the steering angle on the rostopic.

We also created another python file designed to take input from a PS4 controller. We initiated a node in it to subscribe to rostopic - **"/steer"**, converted the value into Integer data type and published it to rostopic - **"e2octrl"**.

7.4 Input to e2o Mahindra Car

This rosoptic - **"e2octrl"** has the message type - **"e2o_ctrl"**. This message is a customized ros message having the following variables -

string RNDB
uint8 Indicator
uint8 Lamp
int8 Steer
uint8 Brake
uint8 Accel
uint8 Wiper
uint8 Horn

Fig 7.1 e2o_ctrl ros message

All the other variables are input from PS4 joystick except Steer. Steer is assigned the value of steering angle which is published by the main python code by rostopic - **"/steer"**. This is done in the file - **"ps4_lane_e2o_ctrl"**.

The car then through Controller Area Network (or CAN) communication linked all of the electronic systems within a car together to allow them to communicate with each other. Through this car moved according to commands given by user and vision code.

Chapter 8: Conclusions and Results

8.1 Conclusion

8.1.1 Lane Detection

We were able to implement a robust lane detection algorithm that could work well even in dim lights and overcast conditions. It can remove all the noise lying outside the road and be unaffected by shadows and lighting conditions.
We also **successfully** handled the discontinuous lane marks, cross sections, missing lane marks, zebra crossings and horizontal lines by applying filters. Using Kalman filter we were able to generate accurate lane markers on curves.

8.1.2 Control System

We successfully implemented a motion planning algorithm to decide the optimal steering angle of the car by comparing the calculated waypoint from the perception model and the heading direction of the car. We made the heading direction determination more robust by sensing the heading direction from the front part of the car. We also implemented multiple techniques such as a bounding box and the average of stored values to minimize wobbling during control. These results were integrated with the car control system for achieving an autonomous lane following system. We were **successfully** able to run the autonomous car for 800 meters stretch without human interference on a path free of shadows.

Some uncertainties involved in the control system include the existence of long continuous paths with only one lane and the response time of the system in case of sharp turns. There is still some wobbling which needs to be reduced further to be able to test at higher speeds.

8.1.3 ROS Implementation

We successfully implemented ROS for integration of the control action with the car control system. The stereo image was detected by the ZED camera and was published to a rostopic, which our code subscribed to and processed the image to determine the steer angle required for correction. We published the steer angle to another rostopic which was subscribed and controlled the car. We successfully integrated the different subsystems using ROS such that the correction takes place without lag.

8.2 Future Work

The primary goal of this project is to attain a fully autonomous car. Some of the problems encountered currently which we plan to solve are -

1) Solving issues of lane detection on U-turns and Sharp turns without lanes. We are trying to integrate GPS with the code so that the car can take these turns using the GPS information rather than unreliable lane detection feedback.

2) Another issue is faced during the trial run when car speed is extremely slow (2-3 km/hr). In such a situation, the shadows and all the obstacles pose a problem as we get a large number of the values with errors because the car is at almost the same position for a long time.

3) The humans on the road also pose a problem. So we are planning to use neural networks to detect humans and all other obstacles and not consider the lines generated due to them.

4) We are also planning to include colour filters to detect white lines to prevent detection of lines due to obstacles and shadows.

References

1. Coursera Course on Self Driving Cars by University of Toronto
   https://www.coursera.org/learn/intro-self-driving-cars
2. Open CV tutorials
   https://docs.opencv.org/2.4/doc/tutorials/tutorials.html
3. OpenCV Lane Detection
   https://medium.com/@mrhwick/simple-lane-detection-with-opencv-bfeb6ae54ec0
4. ROS
   http://wiki.ros.org/
5. Gazebo Tutorials
   http://gazebosim.org/tutorials
6. Pattern Recognition and Machine Learning
   www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf
7. Motion planning
   https://towardsdatascience.com/deeppicar-part-4-lane-following-via-opencv-737dd9e47c96
8. Kalman filter
   https://www.intechopen.com/books/introduction-and-implementations-of-the-kalman-filter/introduction-to-kalman-filter-and-its-applications