Autonomous car following traffic lane & signs

Abinaya Rajesh, Eugeniu Vezeteu, Ke Zhou and Patrick Silatsa
Technische Universität Berlin, DAI-Lab,
Ernst-Reuter-Platz 7, 10587 Berlin, Germany
christopher-eyk.hrabia@dai-labor.de

***Abstract*—Autonomous driving is undergoing huge developments nowadays. It is expected that its implementation will bring many benefits like reshape mobility by enhancing the safety, accessibility and convenience of automotive transportation. For the safety of the passengers, commuters and pedestrians, it is important for the vehicles on the road to follow the traffic rules during the ride. The main goal of the project is to make an autonomous car that follows the road signs and lane. The whole set up of an autonomous car that follows the traffic signs and lane was implemented in a ROS simulation environment. The solution was first proved on a simulation and eventually deployed on an autonomous car kit equipped with Slamtec RPLidarA3, Nvidia Jetson TX2 and Intel RealSense with IMU.**

***Index Terms*—autonomous cars, self-driving cars, traffic sign recognition, lane following**

I. Introduction

Autonomous driving has become a hot topic in the research area of transportation. The advantages of autonomous driving can be enormous. In 2014, there were 32,675 traffic-related deaths, 2.3 million Injuries and 6.1 million reported collisions. Of these, an estimated 94% are attributed to driver errors with 31% legally intoxicated drivers and 10% are distracted drivers [1]. Autonomous vehicles have the potential to reduce the accidents caused by the driver's negligence as the cause of vehicle collisions. They also offer a means of personal mobility for people who cannot drive due to physical or visual disability.

The uses of autonomous vehicles are different in different environments. The private vehicles are the most common in the public, autonomous vehicles are not limited to them. One of the most notable examples in other sectors is the bulk transport of goods. In order to increase the road capacity and the productivity of the companies, a method was developed which is based on the grouping of a number of vehicles as a train. Hence, the autonomous cars and driver assistants systems had brought a huge positive impact to the world.

In this project, a self-driving car that follows the traffic signs and lane has been implemented in the simulation. It has also been tested in a four wheeled car provided to the team. The main goals were to make the cars follow the traffic lane and signs, implement a global planner to generate a path for the car to follow, generate a local planner based on the distance of the car towards the signs and lane detection. The approach was first proved in the Morse simulator before to be tested on a small scaled autonomous car. The environment was unknown, and we had no control over any variable, which allows us to generalize the response of the system.

This paper is structured as follows: In section III, a high level overview of the architecture and some of the methods for their design are presented. The implementation of the algorithms and techniques used in the project are discussed in section IV. The evaluation parameters and the metrics used in the project are discussed in section V. Section VII demonstrates the possible future work that can be done on the project to improve the current performance.

II. Background

For this project, the team was provided with existing packages in ROS and a Hummer scaled car with the properties of the real autonomous car in Morse simulator. The provided packages can be reused or can be created from scratch. The localization package was reused and EKF template was tuned to remove the noise from the odometry. The hector slam was also tweaked to render a better map. Two cars were provided in real time to choose from. The four wheeled car has been chosen because of the dynamics. Its computation platform is composed of NVIDIA Pascal™ Architecture, 256 NVIDIA CUDA Cores, 8GB RAM, 32GB eMMC, Dual core Denver 2 64 bit CPU und quad core ARM A57 complex. Ubuntu 18.4 was used as an operating system and ROS Melodic Morenia was installed. All algorithm and tools used in this project are based on the ROS system. Python was a primary language used in the system.

III. System Analysis and Design

A. Software Architecture

In this section, the architecture of the system is presented in detail. Figure1 gives a high level architecture of the system. Furthermore, sensors that has been used and the relationship between modules are described. The system comprises of several sensors and the sensor that has been used for the purpose of implementation are:

- Camera
- Lidar
- IMU
- Wheel odometry

Perception module is based on Traffic lane and sign detection, Extended Kalman Filter (EKF) and mapping. Decision to go straight or to turn is based on traffic lane and traffic sign. Result of camera perception is fed to Local planner, which is used for trajectory generation. The trajectory is a list of world coordinates what the car should follow. Trajectory generation

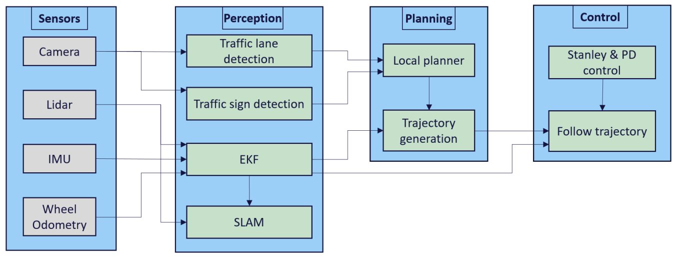


Fig. 1. System architecture

utilize the localisation provided by Extended Kalman Filter and via points given by local planner to compute the world coordinates for path. When the path is generated, motion controller is applied to follow the path. In order to follow the path, Stanley controller has been used to compute the desired angle of the car, and Proportional Derivative (PD) controller to compute desired linear velocity. Spline functions were used to smooth trajectory. In this way, at each step desired world coordinates, and desired speed has been generated. The speed will be reduced at turns and higher speed will be provided in case of straight path. And, EKF is used in this step, to check if the car has reached its destination, based on the Euclidean distance between the car position and the goal.

IV. Implementation

A. A* Algorithms

In this system, A* algorithm [2] has been used to plan the path which was computationally faster when compared to the RRT* and rendered better path. The occupancy grid map which was generated from scanning the environment has been used as an input to the A*. The occupancy grid has values to represent the cells which are occupied and which are left free.

The representation of occupancy grid values:

- 100 - occupied cells
- 0 - free cells
- -1 - unknown cells

A* algorithm is a grid based search algorithm. In the occupancy grid, the cells which are occupied are mapped to 100, the cells which are free are mapped to 0 and the unknown cells are mapped as -1. To make it convenient, the obstacles are mapped 1 and the free cells are mapped to 0. It is optimal to find the path to reach the goal at a minimum cost. The algorithm finds a way through eight of its neighbors to reach the goal. The neighbour through which the cost is minimum is considered and chosen as a next cell through which the car travels. The cost is set in such a way that the cost will be higher(2) when you traverse diagonally and the cost will be lower(1) when you travel straight.

The cost for the robot to reach the goal is calculated based on a heuristics. The heuristics to get the distance between the start and the goal point is calculated based on the Manhattan distance. The path generated was very close to the obstacle which made difficult for the robot to follow the path. Hence, the obstacles were inflated and hence the path generated was

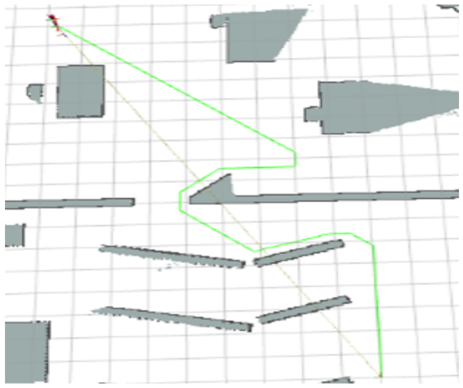


Fig. 2. Path generated from A* algorithm

away from the obstacle making it easier for the robot to follow the path. From fig.2 shows the result generated from the algorithm. The astar algorithm took `two seconds` to generate the referred path. For x=(a,b) and y=(c,d), the Manhattan distance heuristics is given by, $|a - c| + |b - d|$. Though the planner has been implemented, it has not been incorporated with the actual system due to the limited available.

B. Traffic Sign Detection Recognition

In this sub-section the techniques that has used to perform traffic sign detection and recognition is described. The technique is in 2 steps where the first step is *detection* and the second step is *recognition*.

1) Traffic sign detection: As shown in the fig. 3, we assume that our traffic sign are circles, so we reduce the problem of traffic sign detection to circle detection. Mathematically circle is defined as $(x - x_{center})^2 + (y - y_{center})^2 = r^2$, where r is the radius and x_{center} and y_{center} is the center of the circle.

In order to detect circles in image we used opencv hough gradient technique, which uses gradient information of edges, and for each pixel computes the evidence of being a part of a circle. Before applying hough transform, the frame need preprocessing. We convert it to grayscale and applied median blur to avoid false circles and to reduce noise. Detailed explanation of opencv circle detection can be found in reference [3]. Result of this step can be seen in fig 3, detected region is surrounded by red.

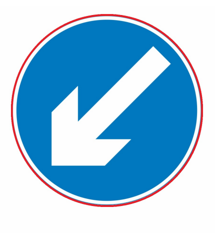


Fig. 3. Traffic sign detection

2) Traffic sign recognition: In order to recognise traffic sign we use deep learning techniques. We created a small image classifier and combined it with circle detection presented in previous part.

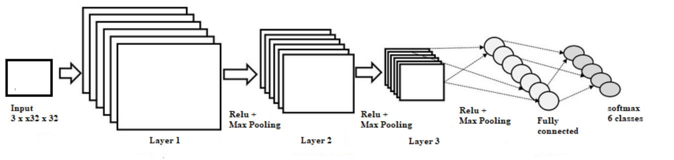


Fig. 4. Neural network architecture

In order to extract feature from the image we used 3 Convolutional Neural Network (CNN) layers, for learning 2 fully connected layer were added. We used softmax in the last layer, to compute the score for each class. To avoid over-fitting we used dropout layer and batch normalisation . The network architecture can be seen in fig.4

In order to solve the problem, several steps were performed:

- **Dataset** - In order to train our model we used German traffic sign dataset provided [4]. It contains RGB images of different sizes, starting with 25x25 to 128x128 pixels.
- **Pre-Processing** - Since input to our model has dimensions 3x32x32, we need to force the training images to have the same dimension. Before training, we forced all images in the dataset to have 3x32x32 dimension.
- **Training** - For training we used model presented in fig.4. We use cross entropy as loss function:
$$L = -\sum y * \log(\widetilde{y}) \qquad (1)$$
We used Stochastic Gradient Descent optimiser and trained our model for 10 epochs.
- **Evaluation** - For evaluation criteria we used k-fold validation, we split our data set in 80% for training and 20% for testing. We obtained 99% accuracy on unseen data.
- **Prediction** - In this step we use circle detection presented in previous part, to detect the region from the image that corresponds to traffic sign, we extract that region and feed our model to get predicted class.

C. Lane Detection

Traffic lane detection is one of the preliminary step involved by autonomous driving cars and it represent the core of our planning package. In this section we will present opencv based algorithm for lane detection.

Below are the steps that we performed in order to detect traffic lanes:

- **Capturing frames** - As input to our lane detection we get frames from the front camera sensor from the car. We used opencv bridge to capture the data from the car and convert it to sequence of images.
- **Convert image to grayscale** - Input frames are in RGB format, we convert them to grayscale because of computational speed purpose (it is easier to process a single channel frame than 3 channel frame), also the colors do not play an important role for this algorithm, because we will detect the change of contrast, not of colors.
- **Noise reduction** - In this step we use Gaussian filter to perform frame smoothing.
- **Edge detection** - In this step we compute the gradient of the image in all directions, and get the edges with higher changes in intensity. Derivatives in x and y directions are computed and then we take larger derivatives as high intensity, smaller derivatives as low intensity. This step was performed using opencv Canny function. More details can be found on reference [5].
- **Segmentation** - In this step we want to extract specific region of interest from the frame, since we are interested in searching of lane only in specific part of the frame. We created a mask with the same dimensions as initial frame and then performed bitwise *AND* operation for each pixel to get our desired region.
- **Opencv Hough Transform** - In this step we used opencv probabilistic hough transform to detect straight lines. More details on hough transform can be found on the reference [6].
- **Averaging** - Our previous step is returning a lot of lines of different length and orientation. The purpose of this step is to filter this lines and to get only desired ones. Firstly, we are not interested in horizontal lines at all, so we remove them from the start. Secondly, we need to know the orientation of the line, to do that we compute its slope. If the slope is negative then we treat line as left one, otherwise if slope is positive we know that the line is on the right side. In the end we compute the average of all left and right lines independently, and we get left and right traffic lane. The vehicle was kept in the middle of the lane.

D. Mapping and Localisation

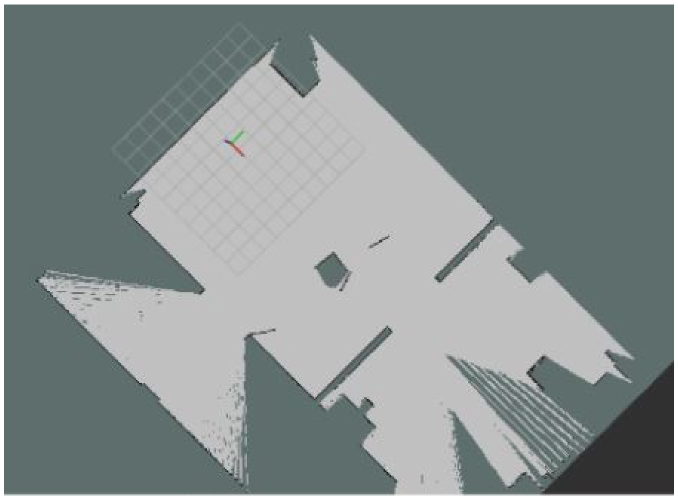


Fig. 5. SLAM map of the environment

The map of the environment and the localization of the car within the map are both based on SLAM. SLAM (simultaneous Localisation and Mapping) addresses the problem of creating a map of the unknown area from a mobile robot while navigating through the environment with the map. The robot starts anywhere in the area and must know at all times where it is currently and the recognized obstacle. The map (see fig.5) used in this project is created by the Hector Slam algorithm. The algorithm uses the Lidar sensor in order to create a map of the whole area, in which the car should drive afterward.

An Extended Kalman filter (EKF) is used to locate the robot within the map and throughout the journey. The Kalman filter itself has 4 components which are linked through equations Eqn.(2) Eqn.(3) whereas

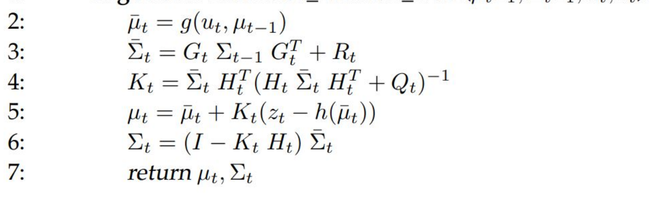
- A_t- is the Matrix (n x n) that describes how the state evolves from t - 1 to t without controls or noise.
- B_t -Matrix (n x l) that describes how the control u_t changes the state from t - 1 to t.
- C_t -Matrix (k x n) that describes how to map the state x_t to an observation z_t.
- $epsilon_t$ **and** $delta_t$ - Random variables representing the process and measurement noise that are assumed to be independent and normally distributed with covariance Q and R respectively.

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{2}$$

$$z_t = C_t x_{t-1} + \delta_t \tag{3}$$

The Kalman Filter assume the linear state transitions and linear measurements with added Gaussian noise, which is so rare in practical. To solve this problem, Extended Kalman Filter utilize the linearization to crack this issue. According to [7] the EKF algorithm is performed by the steps in fig.6. For linearizing nonlinear functions utilize the EKF a method called Taylor expansion [7] . The Taylor expansion constructs a linear approximation to a function g from the value and the slope of g´. The slope is given by the partial derivative in Eqn.(4).

$$g(u_t, x_{t-1}) := \frac{dg(u_t, x_{t-1})}{dx_{t-1}} \tag{4}$$



```
1:    **Algorithm Extended_Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2:        $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3:        $\bar{\Sigma}_t = G_t\, \Sigma_{t-1}\, G_t^T + R_t$
4:        $K_t = \bar{\Sigma}_t\, H_t^T (H_t\, \bar{\Sigma}_t\, H_t^T + Q_t)^{-1}$
5:        $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6:        $\Sigma_t = (I - K_t\, H_t)\, \bar{\Sigma}_t$
7:        *return* $\mu_t, \Sigma_t$
```

Fig. 6. The extended Kalman filter algorithm [7]

The provided EKF combined the information from the wheel odometry, laser odometry and IMU sensors to provide the location of the robot. However, the odometry was not stable due to the noise in the sensors. The yaml file provided with the robot localization package was utilized and the parameters were tuned to get a stable odometry. Process noise co-variance matrix was tuned in such a way that the laser was not noisy more weight was given to the laser sensors. Based on the odometry data, the algorithm can estimate the current position and direction within the map.

E. Obstacle detection

The laser scan which is mounted on the car is of great help which allows to scan the environment and the laser ranges can be used to detect the ranges of the moving obstacles. The laser range between the robot and the obstacle can be measured as the robot moves. When the robot moves closer to the obstacle, the laser ranges become shorted and vice versa. Based on this information, when the laser range becomes lesser than a threshold, we can conclude that the obstacle is near and the decision is taken either to stop the car or to avoid the obstacle. A* global planner is used to map the static obstacles around the car. When a static obstacles are around the car, it can easily take path where there are no obstacles.

F. Local Planner

The trajectory to follow is generated based on the traffic signs and lane detection. When the car starts, it will keep track of the lane and sign with a plan ahead for a certain distance (lets say 3 meters). The local plan is created based on the lane and the traffic sign. When the lane is detected, it will create an array of points to reach the plan ahead distance. When it detect the traffic sign, it will keep track of the traffic signs and takes decision based on the radius of the detected circle in which the traffic sign is enclosed. When the radius of the circle is at a certain threshold (40mm), the car takes decision to turn right or left based on the recognized traffic sign. Hence, the local plan to be followed is generated based on the traffic signs and lane. The next step is to follow the local plan and it is explained the coming sections.

G. Trajectory Generation and Following

Based on the traffic signs and the lane, a smooth the trajectory has been generated by using the cubic spline function. Then, this trajectory followed by two kinds of controller. First, the Stanley controller is used to produce the desired angle. Second, the PD controller is used to make the robot move at desired velocities.

1) Spline function: The path generated by the local plan has very sharp edges which makes it difficult for the cat to follow the path. Hence, the cubic spline was introduced to smooth the local plan generated and make the path easier for the robot to follow. The second order derivative of the cubic spline is used in our project. The second order derivative of the cubic spline is used in our project.

In our project, trajectories are represented as a cubic function like this:

$$s(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \alpha_3 t^3 \tag{5}$$

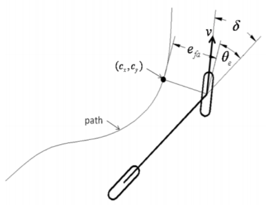


Fig. 7. Stanley method geometry [8]

Here is the way to find the four coefficients of the cubic function,
Equation for the position: $s(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \alpha_3 t^3$
Equation for the velocity: $\dot{s}(t) = \alpha_1 + 2\alpha_2 t + 3\alpha_3 t^2$

By inserting four boundary conditions to get four equations, it's easy to get the values of the four coefficients.
Choose t=0 for those 2 equations:

$s_i = s(0) = \alpha_0$
$\dot{s}_i = \dot{s}(0) = \alpha_1$
Notes: the letter "i" stands for initial

This reduce the equations from four unknowns to two. By insert those value into the original equations, the equations become:

Equation for the position: $s(t) = s_i + \dot{s}_i t + a_2 t^2 + \alpha_3 t^3$
Equation for the velocity: $\dot{s}(t) = \dot{s}_i + 2a_2 t + 3\alpha_3 t^2$

So, the left two parameters now can be solved by only using the other two boundary condition, that 's $s_f, \dot{s}_f, t_f$.
$s(t_f) = s_f = (t_f^2)\alpha_2 + (t_f^3)\alpha_3 + C_1$
$\dot{s}(t_f) = \dot{s}_f = (2t_f)\alpha_2 + (3t_f^2)\alpha_3 + C_2$
Notes: the letter "f" stands for final; C_1, C_2 terms are functions of the initial conditions
Now, arrange those equation into a much more useful matrix formation.

$$\begin{bmatrix} t_f^2 & t_f^3 \\ 2t_f & 3t_f^2 \end{bmatrix} \times \begin{bmatrix} \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} s_f - C_1 \\ \dot{s}_f - C_2 \end{bmatrix} = \begin{bmatrix} s_f - (s_i + \dot{s}_i t_f) \\ \dot{s}_f - \dot{s}_i \end{bmatrix}$$

Thus, the value of a_0, a_1, a_2, a_3 can be calculated. Finally, the trajectory can be generated by using this function.

2) Stanley controller: In this project, the small autonomous car motion is treated as dynamic model. This takes inertial effects into consideration, such as tire slip, steering servo actuation. Stanley control was used to create the right degree of the turn the car should take based on the path generated. The key idea in Stanley control is that the automobile trajectory tracking is treated in a new way, by considering the orientation of the front wheels - not the vehicle's body - with respect to the desired trajectory, enabling collocated control of the system [9]. This approach was used by Stanford university's DARPA grand challenge [10]. Figure7 shows the geometry of the Stanley method. This method is a non linear control mechanism where the heading error and the cross track error are corrected and output the corrected steering angle for the robot to follow. The heading error is given by,

$$\theta_e = \theta - \theta_p \tag{6}$$

In the Eqn.(6), θ is the heading of the vehicle and θ_p is the heading of the path at (cx, cy). The steering angle $\delta(t)$ is given by,

$$\delta(t) = \theta_e(t) + \arctan\frac{ke_{fa}(t)}{v_x(t)} \tag{7}$$

In the Eqn.(7), e_{fa} is cross track error of the vehicle, k is the gain parameter and $v_x(t)$ is the velocity. When the cross track error increases, the steering angle also increases so as to compensate the error.

3) PD controller: The proportional derivative control was implemented to control the speed of the system. The PD controller suffices to produce desired speed for the car to move on the path generated. The P controller is used to reduce the error of the control system. When the constant k_p which the proportional gain factor increases, the error of the system decreases. However, It does not eliminate the complete error in the system. It might add noise to the system as the time changes.

The derivative controller will give response in case of the changes in the system over time. The combination of P and D which is proportional derivative controller (PD controller) provided a better system where the derivative gain factor k_d and k_p are tuned to make the car run in desired speed. The output of the PD controller [11] is given by,

$$u(t) = k_p e(t) + k_d \frac{de}{dt} \tag{8}$$

In Eqn.(8), e(t) is the steady state error with respect to time and $\frac{de}{dt}$ is the derivative of the error term with respect to time. The constants k_p and k_d are the proportional and derivative gains respectively. The values of k_p and k_d are tuned in such a way to produce the better control of the speed of the system.

V. Evaluation

A. Traffic sign recognition evaluation

Our model was trained on German traffic sign dataset provided [4].We trained it for 10 epochs, using 3 different optimisers and cross entropy loss function, training results are provided in Table I.

TABLE I
CNN Training Performance

Optimiser	**Loss**	**Accuracy**
SGD	0.02	0.996
Adam	0.02	0.979
RMSprop	0.72	0.81

We got the best result with Stochastic Gradient Descent optimiser, 0.996 accuracy. Dataset was splitted 80% for training, 20% for testing

B. Motion controller evaluation

Our motion controller is based on low level PD linear velocity and low level Stanley for angular velocity. Based on that, in simulation and real world we had different values for linear and angular velocity. Linear and angular speed limits are presented in Table IV.

TABLE II
LINEAR AND ANGULAR VELOCITY LIMITS

	Simulation	Real car
Linear velocity	1.5	0.5
Angular velocity	(-1.5 1.5)	(-0.5 0.5)

C. CV traffic lane & sign detection

We used OpenCV for traffic lane and sign detection. In this section we specified some tuned parameters for simulation and for real car.
Parameters for OpenCV circle detection:

- **image** - 8 bit grayscale input frame
- **circles** - Output vector of encoded circles as 3D elements $(x, y, radius)$
- **method** - Algorithm used for detection, in our implementation is $CV\_HOUGH\_GRADIENT$
- **dp** - Radio of the accumulator resolution
- **minDist** - Minimum distance between the centers of each circle.
- **param1** - Higher threshold for $CV\_HOUGH\_GRADIENT$ method
- **param2** - The accumulator threshold for $CV\_HOUGH\_GRADIENT$
- **minRadius** - Minimum radius of circles
- **maxRadius** - Maximum radius of circles

Tuned parameters for circle detection are provided in Table III

TABLE III
CV CIRCLE DETECTION TUNED PARAMETERS

	Simulation	Real car
method	$HOUGH\_GRADIENT$	$HOUGH\_GRADIENT$
Radio	1	1
minDist	8	500
param1	100	100
param2	30	50
minRadius	1	20
maxRadius	50	200

Parameters for OpenCV line detection:

- **image** - 8 bit grayscale input frame
- **lines** - Output vector of detected encoded lines, as (x_1, y_1, x_2, y_2)
- **rho** - Distance resolution of the accumulator
- **theta** - Angle resolution of the accumulator
- **threshold** - Threshold for accumulator, keep only viable lines
- **minLineLength** - Minimum line length
- **maxLineGap** - Maximum gap between points on the same line

Tuned parameters for line detection are provided in Table IV

TABLE IV
CV LINE DETECTION TUNED PARAMETERS

	Simulation	Real car
rho	3	1
theta	$\pi/180$	$\pi/360$
threshold	50	10
minLineLength	5	70
maxLineGap	20	20

VI. CONCLUSION

In this Project a fully autonomous driving model car is implemented with ROS as operating system. The whole navigation of the car is based on recorded SLAM map, traffic lane and traffic sign detection and recognition . The car is able to navigate through an area with the help a camera, LIDAR sensor and the map. It can successfully reach a desired goal in the map by tracking the traffic sign and lane. The map and the Localization are based on SLAM. Both are done using the hector slam Library for ROS. Traffic sign and Lane Recognition is done with convolutional neural networks for feature extraction. The navigation part of the car is divided into two different Planner. One is the global planner to calculate the shortest path from an estimated starting point to a set goal. The calculation is based on all charted obstacles within the recorded map and the A* algorithm. The second planner is the local planner to find the best route at the current incidents. It will replan the local route if the LiDAR sensor will detect uncharted obstacle. It will also replan the path if it isn't possible to follow the path without ranking. The local planner generates the velocity commands too.

VII. FUTURE WORK

Future works can be carried out to improve the performance of the existing system. Below are the few improvements that can be done,

- Currently, the car can drive autonomously from the starting point to its destination, but if an uncharted obstacle appears or if there are some obstacle close enough to the car, it will stop. In future, an alternate local plan will be generated to avoid the obstacles and for the car to follow rather than stopping the car.
- Though the A* algorithm has been implemented and tested successfully in the provided environment, it has not been incorporated with the actual project for the robot to follow due to the time constraints to test the code in the real car. A* algorithm can be added to the project to make the robot to follow the path to avoid the static obstacles and it can be used by the car to follow the path in the second round.
- A lane detection algorithm has been implemented successfully to detect the lanes. However, the lane detection

fails sometimes when the lane is very curved. An alternative approach can be implemented to overcome such failures.

References

[1] "2014 crash data key findings," in *TRAFFIC SAFETY FACTS, NHTSA*. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812219

[2] "Direction based heuristic for pathfinding in video games." [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050915004743

[3] OpenCV. Hough circle transform. [Online]. Available: https://docs.opencv.org/3.4/d4/d70/tutorial_hough_circle.html

[4] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The german traffic sign detection benchmark," in *International Joint Conference on Neural Networks*, no. 1288, 2013.

[5] B. Green. Canny edge detection in opencv. [Online]. Available: https://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html

[6] OpenCV. Hough line transform. [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html

[7] "Probabilistic robotics." [Online]. Available: https://docs.ufpr.br/~danielsantos/ProbabilisticRobotics.pdf

[8] "Automatic steering methods for autonomous automobile path tracking," in *CMU*. [Online]. Available: https://www.ri.cmu.edu/pub_files/2009/2/Automatic_Steering_Methods_for_Autonomous_Automobile_Path_Tracking.pdf

[9] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *2007 American Control Conference*. IEEE, Jul. 2007. [Online]. Available: https://doi.org/10.1109/acc.2007.4282788

[10] S. A. I. Laboratory. Stanley: The robot that won the darpa grand challenge. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-73429-1_1

[11] F. A. Salem and A. A. Aly, "PD controller structures: Comparison and selection for an electromechanical system," *International Journal of Intelligent Systems and Applications*, vol. 7, no. 2, pp. 1–12, Jan. 2015. [Online]. Available: https://doi.org/10.5815/ijisa.2015.02.01