nappAI's Lane Following AV Pipeline

Neeloy Chakraborty
Electrical and Computer Engineering
University of Illinois
neeloyc2@illinois.edu

Abhi Kamboj
Electrical and Computer Engineering
University of Illinois
akamboj2@illinois.edu

Pranav Sriram
Electrical and Computer Engineering
University of Illinois
psriram2@illinois.edu

Pavitra Shadvani
Electrical and Computer Engineering
University of Illinois
pavitra3@illinois.edu

***Abstract*—Lane following is a crucial component of self-driving cars, contributing to both the safety and the comfort of the experience. However, creating a robust lane-following system that can operate in unfamiliar environments is a difficult problem. Existing methods primarily use gradient and light thresholding for lane detection and advanced global positioning and localization techniques for lane following.**

We propose a pipeline with a similar lane detection module that implements light and gradient thresholding techniques and a lane following module that dynamically generates the desired way-points for the vehicle to follow. A PD controller is then used to generate the low-level action based on the current pose of the vehicle and the generated desired way-point. The AV pipeline was tested on a simulated GEM vehicle in a Gazebo environment.

For quantitative results, we evaluate the safety and quality of the system using two metrics: average distance travelled and total time taken. We ran three separate trials in the simulator (with different starting coordinates) and measured these metrics to analyze the performance and efficiency of our pipeline. We also compare the path taken by a PD controller using hardcoded way-points for the simulated track and the path taken by our system to observe a mismatch in performance. Experiments run show that our system fails primarily on straight roads whenever the lanes are failed to be detected. Our image filtering method must be further tuned to better detect lanes.

***Index Terms*—autonomous vehicle, image filtering, vehicle controller, ROS**

I. Introduction

A. Motivation

One of the most important aspects of self-driving cars is the safety of the system that many different components of an autonomous system contributes to. The car has to know when to brake and how to avoid obstacles. However, one area that is often overlooked in the safety of the system is its need to follow lanes. Cities and municipalities intentionally marked the lanes of the road for the purpose of keeping its citizens safe and its system in order. Following lanes not only assures that the car is following the rules of the road, but it also increases the chances of a safe drive. A car that cannot stay in its lane is much more likely to face obstacles or pedestrians, which increases the probability of the braking/avoidance system failing. This in turn puts many more people at risk. In order to keep a car in its lane, there are two overarching systems that are running. One is the lane detection system that uses computer vision algorithms on the RGB camera to help the system plan where to go, and the second is the planner that guides the system to follow the path.

In addition to providing order and safety in autonomous systems, lane following provides a much more comfortable driving experience for the users, psychologically and physically. Hypothetically, in a society where there are no lanes, even if cars are able to successfully get their passengers to the correct destination, they would probably not feel safe at all times. The user needs to understand to some extent what is going on with the system for them to feel comfortable using and being around it. Especially since people in society are used to driving the car themselves by following the lanes and rules, they would psychologically feel unsafe sitting in a car where they can't predict where it is taking them. Furthermore, the user might not be physically comfortable sitting in a car that does not have good lane following abilities. This aspect has more to do with the controller, as a PID controller often has the ability to make the transition back to a path much smoother when the controller is critically damped. A safe and comfortable ride is what a user needs for an autonomous vehicle to be considered successful.

B. Challenges

The main challenge for lane following in autonomous systems is the robustness of the system. Lanes can be so different in so many different scenarios and the sensor inputs can also vary given the environmental conditions. Some roads and highways are very well and consistently marked, while others are partially marked, and some are barely marked at all. Lane following systems have to deal with all of these conditions and be able to keep the car on track. Furthermore, weather can often affect these situations as well. When it snows, the lanes are often covered and the car must still be able to determine how to drive. Lane detection algorithms often use lightness threshold to determine where the lanes are, however, this would not work well for snow, since snow might cross this threshold. Any vertical patches of snow on the road, such as tire markets may be seen as lanes to the vehicle. Adapting to

all these different conditions makes lane following a difficult challenge that is still being researched and developed today.

Another challenge with lane following is characterizing the inputs and references to the controller system. In simulations, autonomous vehicle systems often use hardcoded way-points within a map. However, when translating this system to a real life scenario, it becomes hard to use these way-points because the autonomous vehicle has to have detailed and precise knowledge of the car and environment. This includes its exact location and the location of all the way-points. This can be done using GPS coordinates or other positioning systems such as SLAM and other odometry techniques along side a designer who provides way-points, however, it is often not very accurate for the level of precision an autonomous vehicle system. The GPS and odometry measurements would need to be extremely robust to noise and work in all different types of conditions and situations. The odometry could be messed up by hydroplaning of the car or skidding on icy roads and the GPS can easily become inaccurate in remote locations or tunnels. Overall, positioning a car on the road and then planning way points for the low level controller to follow is a very challenging task.

C. Existing Methods

Current methods of lane detection use computer edge detection techniques to follow the lanes that exist, and many different advanced techniques are used to correct for them. For example, edge detection techniques often use gradient and light thresholding techniques to detect edges, however, with shadows, reflections, rough weather conditions, etc., the image containing the lanes can often be distorted. Quality and illumination enhancement techniques like sharpening techniques such as adaptive bilateral filter (ABF) are helpful in improving the system [1].

As for generating or mimicking way-points in autonomous systems, learned models using neural networks are often needed to account for the noise in the system and positioning. Some research has been done regarding using GPS and positioning based on previous knowledge of the map, however, other research studies methods to primarily use perception sensors without global positioning. Ideally a car should be able to localize its position given its surrounding and determine where to go, even given no prior knowledge. Training neural networks and decision processes to make such high-level decisions has allowed researchers to combat some of the existing noise problems in localization techniques but the research is still ongoing.

D. Justification

The method we used for lane following attempts to overcome the challenges that arise from having to hardcode way-points in a map. We propose and evaluate a method for dynamically creating way-points relative to the car's current position to feed into a proportional-derivative (PD) controller based on image data. This removes the issue of a user having to create way-points themselves, but in a real-life scenario, the car may still face many other difficulties with noisy inputs. Our approach does not handle the lane detection very well in various conditions as we simply combine gradient filtering and color thresholding for line detection. However, since the simulation always has well marked lanes, and we receive an accurate pose estimation of the vehicle at all times from Gazebo, it makes for a stable environment to test the dynamic way-point method we implemented. By using a stable lane detection algorithm and only focusing on the image transformations and way-point generation, we are able to experiment and develop a much more robust system than if we had spread the project thin and tried to focus on too many aspects at once.

E. Literature Review

The concept of autonomous vehicles has been studied for the past couple decades and various surveys on lane detection highlight the common trends in challenges and advances amongst numerous papers. Zhu et al. provide a comprehensive overview of perception for intelligent vehicles [2]. They describe various sensors that contribute to the perception system of the autonomous vehicle. When discussing lane detection, they highlight the various applications of the systems. Lane detection is not only useful in autonomous systems but also is used in current vehicles in lane departure systems, adaptive cruise control and lane keeping systems [2]. However, since these are all assistance systems, they do not require the algorithm to be as robust as a level 5 autonomous vehicle. VKastrinaki et al. highlights the many difficulties in creating fully autonomous systems, including noise from 3D reconstructions in the scenes and human errors. There are various levels of noise that comes from the different layers of autonomous vehicle system. High level decision making algorithms not only need to account for the fact that the hardware sensors may face noise, but also that the algorithms that perceive data from the hardware are often learned models that are not perfect and may create inaccurate models of the world [3]. Furthermore, human and environmental errors such as a gust of wind accidentally nudging the lidar or camera can also influence the system. This is extremely difficult to account for since it can't be predicted and fixed like gaussian or salt and pepper noise for images. Feniche et al. breaks down lane detection algorithms into five steps: image cleaning, feature extraction, model fitting, time integration, and image to world correspondence [1]. They further survey techniques used in research for each step, such as using thresholding and smoothing for image cleaning, using the Canny Edge operator or Symmetrical Local Threshold for features extraction and so on and so forth. Duong et al. uses some of these techniques to create a fully functioning lane detection system [4]. This system warps the image perspective to a birds eye view, performs lane detection, edge detection and lane fitting, and warps the image back with the detected lanes. Our project implements a very similar system to Duong et al. with a few minor differences, such as using a Sobel filter and a parabolic function for edge detection instead of a Hough transform.

Along with lane detection, much research is focused on how to integrate lane detection with the system. Li et al uses a fusion of vision and lidar sensor data to map out the optimal drivable region on a road [5]. They then train a model that conditionally uses lane detection if the lanes fall within this optimal drivable region. This system is designed to account for inconsistencies in the lane markings and implementing such a learned model for decision making onto the dynamic way-point following system we implemented in this project would allow our system to be more robust to errors.

II. Methods

This section describes each of the modules of our proposed lane following autonomous vehicle and the implementation process in simulation.

A. Simulator and GEM Vehicle

Our pipeline was developed using ROS and Python to work on the GEM vehicle in a Gazebo simulation whose environment holds a closed-loop track with two lanes and several turns. The simulated GEM vehicle is equipped with several sensors including an RGB camera on its dashboard, a 360° Lidar on its roof, a front Radar, and a pose estimator. In this project, we use the RGB camera and pose estimator as input sensors to plan dynamic way-points that our controller uses to produce low-level actions, as described in later sections. The vehicle is able to take in steering wheel angle commands and a linear velocity command to move the vehicle towards a desired direction with a given velocity.

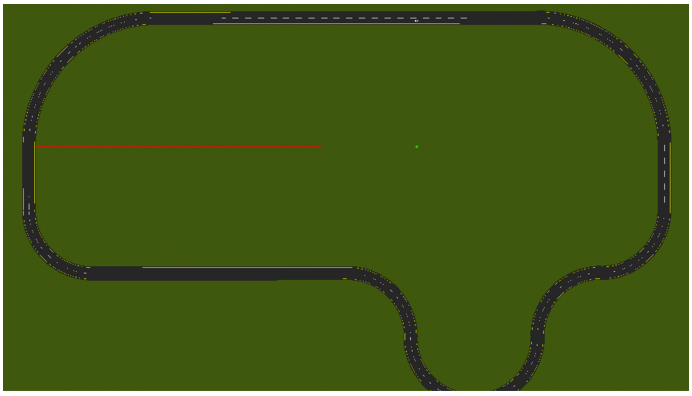

Fig. 1. The track that the GEM vehicle drove around in the Gazebo simulator.

B. Overall AV Pipeline

The nappAI lane following pipeline consists of five main sub-components: an information gathering module that takes sensor values from the environment, an image processing unit, a planner, a controller, and an output bridge to send commands to our vehicle. At every time step, the information gathering module will query the AV's dashboard camera for an image frame and the GPS for pose information. The image frame is processed to identify the location of points on the image that represent five meters in front of the car to stay within the lane. Our planner converts the pixel space dynamic way-point to a point relative to the base frame of the car. The controller then converts that base frame point to a world frame point and produces a low-level action to decrease the error in the current pose and the desired way-point. Finally, the low-level control actions propagate the vehicle forward and the cycle of the pipeline begins anew.

C. Information Gathering and System Inputs

The Gazebo simulator publishes data to four relevant ROS topics for the image and vehicle pose information. Specifically, we require the RGB camera's intrinsic properties that are published to the *front_single_camera/camera_info* topic to later convert from the pixel to world space, and the RGB frames themselves that are sent to the *front_single_camera/camera_info* topic. Though, we must ensure that these two data points are received by our image processing unit at similar times. This constraint is guaranteed by filtering data packets through a synchronizer.

Vehicle pose information is gathered from the */gazebo/get_model_state* topic which provides a world position and orientation in quaternion form. Finally, ROS provides the *tf.TransformListener.transformPoint* method to convert points between various coordinate frames in the world. These different topics and modules allow the nappAI pipeline to take in image data, produce dynamic way-points, and convert those way-points from one frame to another to work with our low-level controller.

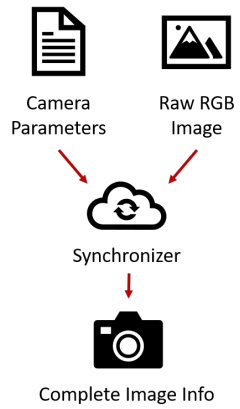

Fig. 2. A diagram of the process to synchronize camera intrinsic properties and RGB frame.

D. Image Processing

An image processing unit combines several filtering techniques to detect lanes and generate way-points in the pixel space of the RGB dashboard camera. Given an input RGB image, the picture is processed with color thresholding and finite difference filtering in two different pipelines and then combined to emphasize the pixels denoting lanes (filtering was done through OpenCV). Specifically, color thresholding filters pixels that are yellow and white because the lane markers are yellow and white respectively. Finite differencing in the form of a Sobel filter was applied to a blurred grayscale version of

the image to find edges. Combining the outputs from these two filters together provides a clearer understanding of which pixels in the dashboard RGB image represent lanes.

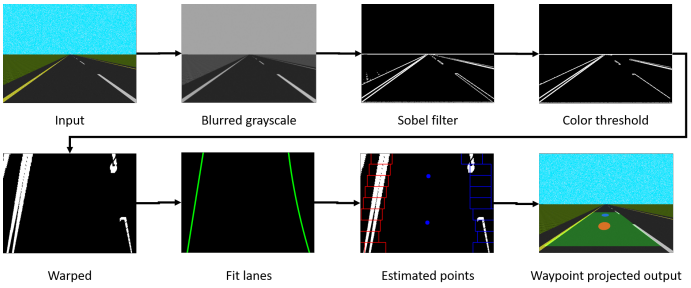

Fig. 3. Steps used in image filtering unit.

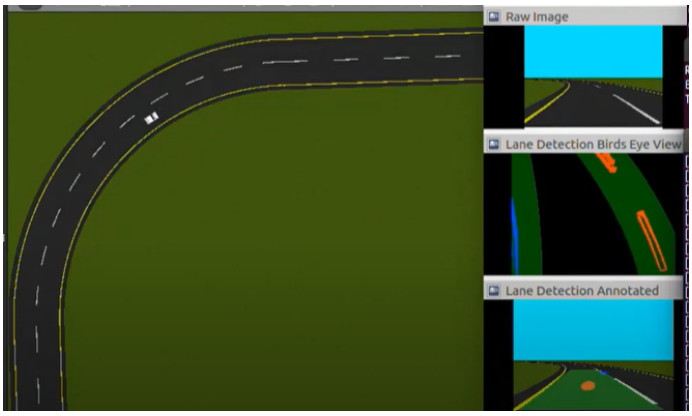

Fig. 4. A view of projected detected lanes and generated way-points.

Given this image, we crop out a section of the image that focuses on the lane in front of the vehicle and produce a new top-down warped image of the lane using OpenCV's perspective transform methods. This warping step makes the next lane detection algorithm perform more accurately by reducing the chance of detecting outside pixels as a lane. The lane detection algorithm identifies high-intensity pixels that are near the left and right edges of the image by using a sliding window method. Specifically, windows are generated from the bottom of the image to the top based on the estimated heading of the lanes. The non-zero pixels within these windows are expected to be lanes and are used to fit cubic polynomials for both sides.

With an estimated pair of polynomial fits of the heading of both lane edges, we generate a point that is 330 pixels down from the top of the image and centered at the middle of the two lanes at that height. This point represents a distance of five meters away from the vehicle. We then un-warp the top-down image to be projected on the full dashboard image input along with the five meter way-point. This step then concludes the purpose of the image processing unit: given an input RGB dashboard image, we create a point in the pixel space that represents the estimated location of the center of the lane five meters away from the GEM vehicle.

E. Planner

To produce a low-level control action, our PD controller uses the error between the current position and heading in the world frame to the desired location. For that controller input to be in the correct format, our planner first converts the generated way-point from the pixel space of the camera to the base frame of the car. First, the planner converts the pixel-space coordinate to the camera frame coordinate space by making use of the fact that the camera may be represented by the simple pinhole camera model. Given a pixel coordinate (u, v), the focal point f, and height Y of the camera in the world, we can find the relative horizontal distance X and depth Z of the pixel point in the real world relative to the camera frame according to (1)-(3).

$$\frac{f}{Z} = \frac{u}{X} = \frac{v}{Y} \tag{1}$$

$$u = \frac{fX}{Z} \tag{2}$$

$$v = \frac{fY}{Z} \tag{3}$$

This calculation is simplified by using the ROS *image_geometry* module, which has methods to take in the intrinsic camera parameters and pixel coordinates, and generates a unit vector in the direction of the camera frame point. We scale this unit vector such that the height Y matches the actual height of the camera, and we now have a 3D way-point relative to the camera frame.

As stated earlier, we must now convert the camera frame point to the base frame before the controller can use the point to generate a low-level action. The ROS *tf* module was used to convert the camera frame way-point to the base frame by performing the simple transformation in (4). This base frame coordinate is published every time step to a topic that the controller is subscribed to.

$$P_{base} = T_{base,camera} \cdot P_{camera} \tag{4}$$

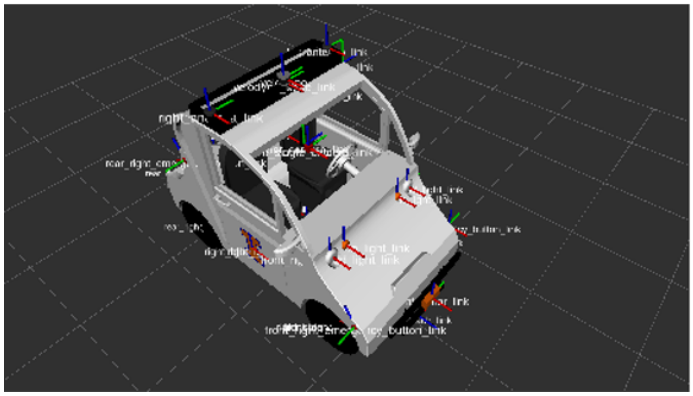

Fig. 5. The GEM vehicle and all of its transformation coordinate frames.

F. Controller

The final transformation we must perform before the controller may act, is from the base coordinate frame to the world frame. There is no direct transformation matrix that exists in simulation to convert between these frames, so we applied classical geometry methods to this task. Recall, the controller is given a point (x', y') from the planner in the base frame of the car, and the yaw θ of the vehicle relative to the world frame is found from the model state publisher. Then the distance h' to the base point in the base frame and the relative angle ϕ between the x-axis of the car and point may be found as follows:

$$h' = \sqrt{(x')^2 + (y')^2} \tag{5}$$

$$\phi = \arctan(\frac{y'}{x'}) \tag{6}$$

These metrics are then used to calculate the projection (x'', y'') of the vector to the way-point in the base frame on the world frame:

$$x'' = h' \cos(\theta + \phi) \tag{7}$$

$$y'' = h' \sin(\theta + \phi) \tag{8}$$

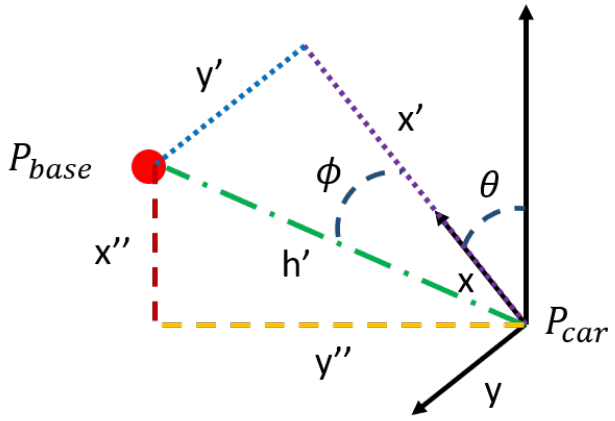

Fig. 6. Visualization of geometry used to convert from base frame to world frame.

Finally, we can add those projected distances to the current relative position of the vehicle to the world frame to produce a way-point in the world frame. This new way-point is used to calculate the error $[\delta_x, \delta_y, \delta_\theta, \delta_v]$ between the current state $[x_B, y_B, \theta_B, v_B]$ and the reference state $[x_{ref}, y_{ref}, \theta_{ref}, v_{ref}]$ according to (9)-(12) [6].

$$\delta_x = \cos(\theta_B) \cdot (x_{ref} - x_B) + \sin(\theta_B) \cdot (y_{ref} - y_B) \tag{9}$$

$$\delta_y = -\sin(\theta_B) \cdot (x_{ref} - x_B) + \cos(\theta_B) \cdot (y_{ref} - y_B) \tag{10}$$

$$\delta_\theta = \theta_{ref} - \theta_B \tag{11}$$

$$\delta_v = v_{ref} - v_B \tag{12}$$

Our PD controller produces a 2D action output with a steering wheel angle and linear velocity values by multiplying the following gain matrix with the current error:

$$K = \begin{bmatrix} k_x & 0 & 0 & k_v \\ 0 & k_y & k_\theta & 0 \end{bmatrix} \tag{13}$$

G. Low-level Action Outputs

The controller output is published to the */ackermann_cmd* topic with steering wheel angle and linear velocity values. Gazebo completes our lane following pipeline cycle by taking the action on the vehicle in the physical environment.

H. Assumptions and Safety Guarantees

By implementing the lane-following pipeline in simulation, we inherently introduce several assumptions that we would have to reconsider if we were to port this work to a real-world vehicle. First, the Gazebo simulator provides completely accurate vehicle model pose information every time step, whereas we would have imprecise GPS or particle filtering localization in the real world. Our environment tracks have no other humans or vehicles on the road, so we assume there are no other sources for unsafe actions to occur in the simulation. In our design methodology, we also assume that a point five meters away from the dashboard camera is representative of the direction that the vehicle should be heading towards to stay within its lane. Furthermore, we rely heavily on the quality of the image filtering unit to produce top-down lane images from which lanes should be able to be detected at all times. If the lane is not detected at some time, we expect that the system will fail at staying within its lane because the planner will be outputting an irrelevant way-point for the controller to act towards.

In that same vein, our reliance on an image filtering unit that impacts the complete decision making channel makes it difficult to purely use formal verification to quantify the safety of our system. The images received from the RGB camera has added noise according to the intrinsic properties of the camera, and the vehicle standing at the same pose for multiple time steps may produce different filtered lane predictions. This outcome hinders our ability to apply formal safety methods like the Safety Guarantee Automata to our system because we cannot enumerate all possible states efficiently. Instead, we may apply statistical safety approaches, like the test matrix approach, to identify the quality of the system in various states over several runs using actual data from the simulator. Overall, we define our system as acting safely when all four of the AV's wheels are within the lane, the AV is facing the correct direction for the lane it is driving in, and it is driving forward to stay in its lane.

III. Experiments

To evaluate the safety and quality of our system, we ran the system at three different starting locations on the track for three separate trials. Specifically, we start the vehicle in the simulation at coordinates $(120, -98.5)$ to simulate a straight lane, $(200, -98.5)$ for a long left turn, and $(300, 15)$

TABLE I
DISTANCE AND TIME TRAVELLED SAFELY FROM VARIOUS START POINTS

		Trial					
		1		2		3	
		Distance	Time (s)	Distance	Time (s)	Distance	Time (s)
Experiment	Long Left	169.03	820	168.87	817	169.13	824
	Sharp Left	118.86	577	118.66	577	118.89	579
	Straight	39.64	192	40.02	193	39.71	193

for a sharp left turn. On a quantitative stand, we measure the average distance travelled across all three trials for each starting location before two wheels of the vehicle have left the lane. We also take note of the time taken to complete each run. Qualitatively, we will show paths taken by the vehicle from each starting location and compare them against hardcoded way-points that would have taken the vehicle safely through the track. An ideal AV system following this pipeline would be able to reach the full distance of the track from any starting point.

Several smaller design decisions were made to allow our system to drive on the simulation track. On the image processing end, a Gaussian blurring filter with a kernel size of 5, Sobel filters of kernel size 3, and a color thresholding filter for values between $(70, 255)$ were used. The height of the camera from the road was set to approximately 1.546 meters and the target velocity of the vehicle controller was constantly set to 3 m/s. We also found that the controller performed better with fewer oscillations if the generated way-point in the base frame was normalized to length 1 before being converted to the world frame. The gains used for the PD controller were $k_x = k_y = 0.1$, $k_v = 0.5$, and $k_\theta = 1.0$. Finally, a tunable parameter we found impacted the quality of performance of our system was the rate at which data was published to the controller. Too long or short a frequency made the react too late or very fidgety. This parameter had to be tuned differently across different machines with varying computing capabilities.

IV. RESULTS

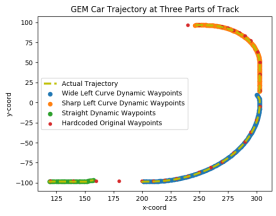

Fig. 7. Trajectories from three start points compared against hardcoded way-points.

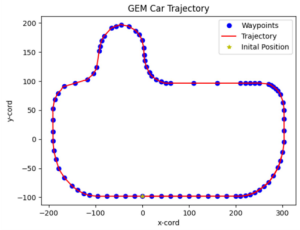

Fig. 8. Trajectory around track from PD controller using hard-coded way-points.

Table I summarizes the distance travelled and time taken from the three start points stated earlier before the car left the track. We can see that running multiple trials from the same start point does not produce a great difference in distance travelled nor in computation time. This observation can be attributed to the fact that the Gazebo simulator was not set

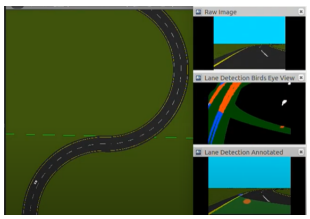

Fig. 9. System beginning to fail to detect lanes and producing skewed way-points.

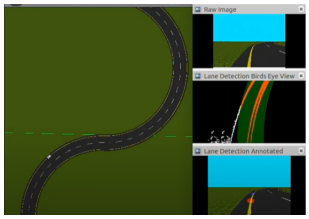

Fig. 10. System turned into line-following vehicle after receiving poor way-points.

to introduce great amounts of noise during execution. Furthermore, we can see that the vehicle travelled the shortest distance on the straight path before failing to meet safety guidelines. On the other hand, as seen in Fig. 7, the vehicle was able to safely complete the two curves tested, but failed immediately after entering a straight. From a qualitative standpoint, this occurred because the lane detection algorithm was unable to fit accurate lines to estimate the curve of the lanes and generated unreasonable way-points. An example of this issue is shown in Fig. 9-10.

We also consider the runtime of the algorithm an important metric because AV systems must run at a high frequency to react quickly to unforeseen circumstances. The data collected for the runtime was taken while the RVIZ visualizer was open (which may have increased the runtime compared to if RVIZ were not running). We find that the system in simulation runs at about 0.2 m/s real time and the majority of computation costs is exhausted by the lane detector. Overall, when visualizing failure cases, we can see that our image filtering method removes far too many pixels that are actually lanes before applying the lane detection method. We visualized the performance of a PD controller with hardcoded way-points in Fig. 8 and find that the controller is able to stably complete the track from its start point. To achieve a similar safety performance on our algorithm, we suggest several steps to improve our system.

The filtering algorithm must be better tuned for the Gazebo simulator to get more accurate results for our pipeline. Furthermore, the computation cost of the lane detection algorithm itself is very high, and we must consider how to make the pipeline more efficient before applying to a real vehicle. Finally, variances in image inputs are inevitable, and disturbances will be applied to the car in the real world. For those reasons, we may consider merging an anomaly detection algorithm to find when the current state of the vehicle is already off the track and produce actions to correct itself. We would also like to run the pipeline on a more diverse set of start points, on multiple different tasks, with increased noise to produce a more representative sample of the performance of our method and develop a more robust image filtering method.

V. DISCUSSION

This paper presented a lane following pipeline applied to a Gazebo simulation GEM vehicle on a closed-loop track.

The nappAI lane following pipeline consists of an input information gathering subsystem, an image processing unit, a planner, a controller, and an output bridge for low-level actions to the vehicle. We tested our pipeline on one track from three start points and evaluated the distance and time the vehicle was able to travel before it shot off the track. A successful system, like a tuned PD controller with hardcoded way-points, was shown to be able to successfully complete the track. In contrast, our method is hindered by the poor performance of our image filtering algorithm that led to inaccurate fitting of lanes, and ultimately produced unreasonable way-points. Interestingly, the filter performs the worst when the vehicle is on a straight road, which we assumed would be the most successful scenario.

There were several challenges faced in the development of our system. The greatest challenge was tuning the image filtering parameters, controller gains, and ROS frequency of publishing generated way-points. Lane detection itself is a very costly computation for our current implementation, which proved to be a bottleneck in runtime. Furthermore, the GEM vehicle has several transformation frames and we had to ensure the points that were being converted from one frame to another were ordered in the order expected by the frames.

In addition to the proposed improvements in section IV, we suggest future work here. In the real world, pedestrians crossing and other vehicles on the road must be considered alongside lane following. Our simulation could be more realistic by introducing those obstacles and incorporating LIDAR as a method to dynamically control the desired speed of the vehicle depending on detected obstacles. After the method performs better in simulation, we would like to apply the pipeline on a real world GEM vehicle.

VI. Link To Video

Our project video is the same as our final presentation. The mediaspace link is $https : //mediaspace.illinois.edu/media/t/1_0ei81sqa/157425081$ and our presentation starts at 1:07:30.

References

[1] M. FENICHE and T. MAZRI, "Lane Detection and Tracking For Intelligent Vehicles: A Survey," 2019 International Conference of Computer Science and Renewable Energies (ICCSRE), 2019, pp. 1-4, doi: 10.1109/ICCSRE.2019.8807727.

[2] H. Zhu, K. Yuen, L. Mihaylova and H. Leung, "Overview of Environment Perception for Intelligent Vehicles," in IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 10, pp. 2584-2601, Oct. 2017, doi: 10.1109/TITS.2017.2658662.

[3] V Kastrinaki, M Zervakis, K Kalaitzakis, A survey of video processing techniques for traffic applications, Image and Vision Computing, Volume 21, Issue 4, 2003, Pages 359-381, ISSN 0262-8856, https://doi.org/10.1016/S0262-8856(03)00004-0. (https://www.sciencedirect.com/science/article/pii/S0262885603000040)

[4] T. T. Duong, C. C. Pham, T. H. Tran, T. P. Nguyen and J. W. Jeon, "Near real-time ego-lane detection in highway and urban streets," 2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), 2016, pp. 1-4, doi: 10.1109/ICCE-Asia.2016.7804748.

[5] Q. Li, L. Chen, M. Li, S. Shaw and A. Nüchter, "A Sensor-Fusion Drivable-Region and Lane-Detection System for Autonomous Vehicle Navigation in Challenging Road Scenarios," in IEEE Transactions on Vehicular Technology, vol. 63, no. 2, pp. 540-555, Feb. 2014, doi: 10.1109/TVT.2013.2281199.

[6] M. Jiang, Y. Li, and S. Mitra, "MP 2: Vehicle Model and Control," 2021 ECE 484: Principles of safe autonomy, http://publish.illinois.edu/safe-autonomy/files/2021/03/ECE484_MP2_SP2021.pdf.