EasyChair Preprint

№ 1295

Lessons Learned From Deploying Autonomous Vehicles at UC San Diego

David Paz, Po-Jung Lai, Sumukha Harish, Hengyuan Zhang, Nathan Chan, Chun Hu, Sumit Binnani and Henrik Christensen

July 16, 2019

Lessons Learned From Deploying Autonomous Vehicles at UC San Diego

David Paz, Po-Jung Lai, Sumukha Harish, Hengyuan Zhang, Nathan Chan, Chun Hu, Sumit Binnani, Henrik I. Christensen

Abstract While most autonomous driving efforts reported are directed for general driving and mainly on major roads, there are numerous applications for autonomous vehicles for last mile mobility–from person mobility and mail delivery to flexible recharging of cars in parking structures. Over the last year, we have designed vehicles for the micro-mobility challenge. Our approach was based on adoption of the open source Autoware system. The system was taken as a starting point for the design of a robust solution. Proposed requirements include a robust control design, a shift towards increased use of image data over LiDAR data, handling of a richer set of vehicles / pedestrians in a last mile scenario, and overall system characterization and evaluation. We present an overview of the overall design and the design decisions for construction of vehicles for last-mile delivery.

1.1 Introduction

The Autonomous Vehicle Laboratory (AVL) at UC San Diego's Contextual Robotics Institute has partnered with UC San Diego's Logistics Unit, Police Department, and Mail Delivery Center to develop and deploy self-driving carts on campus. At the moment, two GEM e6 development platforms are being used to perform tests around the campus. During the initial development stage, the vehicles are expected to perform mail and package deliveries at different campus locations in a variety of stochastic traffic and road conditions. In the second development stage, the team will lead the effort on last-mile transportation for campus visitors, students, and faculty by providing on demand mobility that can address heavy campus traffic during peak hours, while reducing the amount of on-campus parking needed.

Over the course of seven months, AVL has implemented software modules for sensing and estimation, planning, controls, and benchmarking. Additionally, state

UC San Diego - 9500 Gilman Dr, La Jolla, CA 92093
Email - (dpazruiz, polai, ssumuka, hyzhang, nchan, chh281, sbinnani, hichristensen)@ucsd.edu

of the art algorithms for localization, mapping and perception have been explored and evaluated from multiple open source communities: Robot OS (ROS) [6] and Autoware.AI. [2]. Although some of those software packages discussed in detail in this paper are robust and work very well for certain applications, they do not generalize and scale, and often require extensive software-hardware integration. The evaluation and development results during the testing phase of the project are presented with proposed modifications to improve performance, flexibility, reliability and scalability in autonomous vehicle systems for micro-transit.

1.2 Technical Approach

Each of the two GEM e6 development platforms used for this study have been retrofitted with a drive-by-wire system and a safety disengagement system[1] that provides an interface for controlling steering angle and angular velocity, brake pedal position, acceleration, and gear shifts using ROS. Each of the units is equipped with six Mako G319 cameras, one VLP-16 Velodyne LiDAR, 16 Bosch ultrasonic sensors, and a Varience VMU931 Inertial Measurement Unit (IMU). Additionally, a Watson DMS-SGP02 Inertial Navigation System (INS) is being tested on one of the units. The development platforms are shown in Figure 1.1.

Initial system integration and development was performed using numerous tools, including Docker containers for modularization, the open-source Autoware stack as a development platform, alongside communication APIs for vehicle-to-system control. While a reasonable degree of autonomy was achieved during this development stage, Autoware's localization and planning pipeline heavily relies on dense point cloud maps and LiDAR technology. As discussed in later sections, this technology may be adequate for smaller scale environments but comes at a great cost as the size of these maps drastically increase–computationally and maintenance wise. As part of an ongoing process, AVL has started developing alternative methodologies that provide equal or greater level of system robustness with the primarily goal of improving scalability and integration as a whole. In this section, we introduce tools and the current state of the system and propose alternatives for improvement.

1.2.1 Containers

ROS is an ecosystem that provides a convenient communication interface for our applications and modules. Although ROS offers this convenience, one drawback lies on package interdependencies and libraries (i.e OpenCV) which often leads to problems compiling source code due to incompatible versions. By isolating the

[1] The disengagement system permits safety drivers to intervene at any time using the steering wheel, pedals or emergency button. During field tests, steering wheel and pedal interventions proved to be necessary due to the immediate response needed in many scenarios.

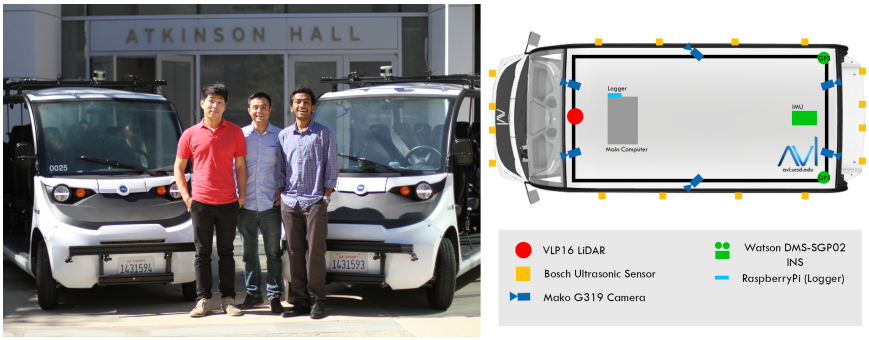

Fig. 1.1 AVL team members and GEM e6 Development platforms are shown on the left. Sensor locations are shown on the right.

different modules into separate environments using Docker containers, this task becomes trivial and allows us to automate the module instantiation process during boot time. UC San Diego's AVs are among the first few platforms to make use of Docker for robotics applications [10].

1.2.2 Autoware/ROS

Two of the most popular open-source autonomous driving stacks with ROS support include Baidu's Apollo and Tier IV's Autoware.AI. Ultimately, Autoware.AI was chosen as the foundation of AVL's autonomous driving stack due to its open source community support and flexibility for modifications and integration. Autoware provides algorithms and packages for perception, mapping, localization, and planning algorithms required for navigation through specific environments. The modules are relatively complete and work well for basic environments, despite some limitations which will be discussed in the latter part of the paper. Several global and local planning modules are also packaged with Autoware, including the A*, lattice, and OpenPlanner, where each planning algorithm is optimized for different scenes.

Out of the different planners provided by Autoware, we chose OpenPlanner [4], which serves as the path planner for autonomous navigation through an urban environent. OpenPlanner is comprised of two distinct modules: a global planner for generating an initial reference path, and a local planner for controlling the behavior of the vehicle. Since AVL's focus is on last mile transportation, our efforts have centered around exploration and modification of the local planner. OpenPlanner supports local planning with a limited number of static road features, such as lanes, stop signs, and dynamic features, such as traffic lights. As of Autoware version 1.8, OpenPlanner lacks support for many interactive road features, such as crosswalks and intersections, which are critical for autonomous navigation in areas with high foot traffic–such as UC San Diego. This entails integration of perception modules

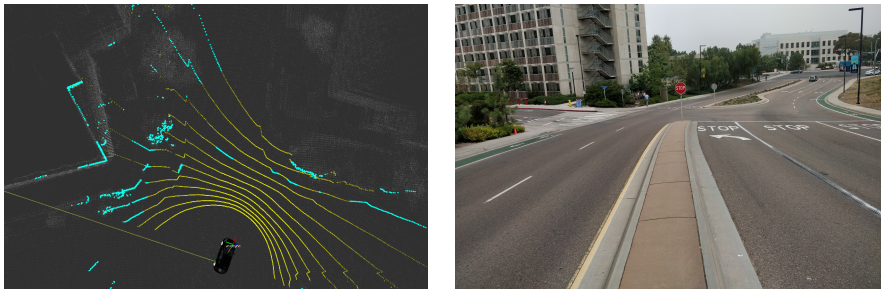

Fig. 1.2 Ground remove failure case along inclines: in the LiDAR scan image (left), yellow points are classified as ground and turquoise points are classified as obstacles. The vehicle is turning left in a three-way intersection and part of the ground is classified as obstacle. The image on the right shows the intersection.

into OpenPlanner and addition of the appropriate behavior logic to the local planner. In addition to missing local planner logic for certain road features, modifications to the perception modules are required to generalize the detection and tracking algorithms for a wider variety of scenes. In the following sections, we will discuss the specifics of the necessary changes.

1.2.3 Perception and Tracking

1.2.3.1 Ground Removal

In the LiDAR detection pipeline, point clouds are projected onto a 2D plane for segmentation and point cloud ground removal to avoid obstacle misclassifications, such as classifying ground points as an obstacle. One proposed implementation is to remove the points that fall below a threshold, or better, to use the Random Sample Consensus (RANSAC) algorithm to fit a plane on the ground. Although these two methods work reasonably well to a certain extent, a failure case is encountered along roads with high degree of inclination as represented in Figure 1.2.

To address ground removal along dynamic road conditions, a simple, but effective technique is to partition the point cloud based on the longitudinal distance to the car and fit a plane in each partition. These planes can provide better plane approximations for curved roads and remove the ground points effectively, as demonstrated in Figure 1.3. The RANSAC plane fitting implementation in the Point Cloud Library (PCL) [9] is initialised with a direction that is approximately normal to the plane in question and generates a normal vector that best characterizes and fits the data, i.e. an angle within a threshold. Using this method, an iterative approach is applied to specify the direction of the planes. First, we assume that the plane that is closest to the vehicle is nearly perpendicular to the z-axis in the LiDAR frame and use this direction as input to determine the actual parameters of the plane by RANSAC. Based

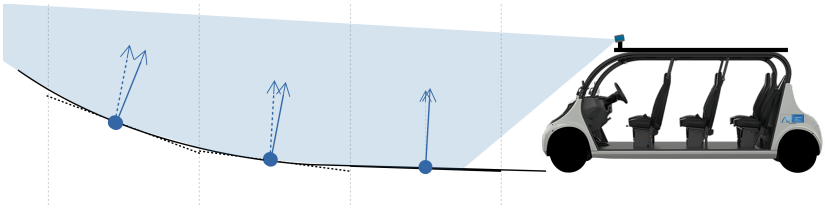

Fig. 1.3 The direction of dashed arrows indicate the RANSAC initializations and the solid line arrows correspond to the normal vectors of the estimated plane.

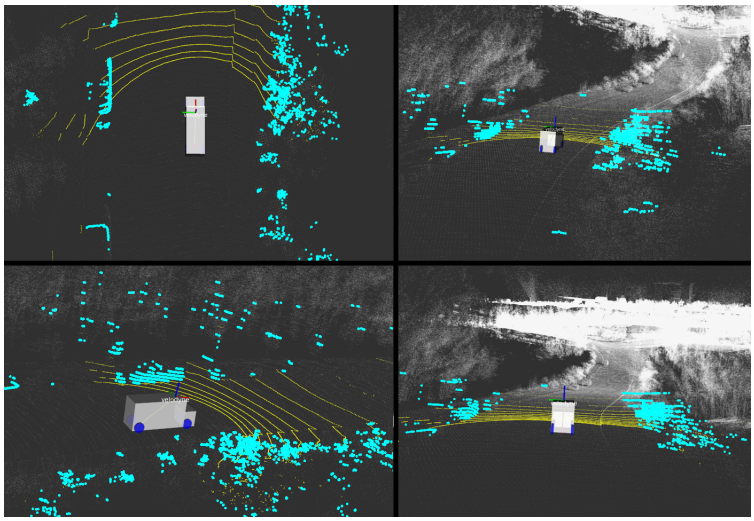

Fig. 1.4 Successful ground classification along an inclined road from multiple perspectives. The yellow scans are classified as ground and are removed. Turquoise scans are considered as potential obstacles.

on the assumption that the slope of the road changes slowly among adjacent partitions, we use the direction of the normal vector of the last estimated plane as the input of the estimation of the next plane.

In Figure 1.3, we separate the road into three segments and estimate from right (the closest partition to the vehicle) to left. This approach works effectively even when the road is curved as shown in Figure 1.4, but the assumption that each partition can be approximated by a plane might not always hold when the change in slope varies significantly within short distances. A similar approach for iterative elevation calculation is considered in [8].

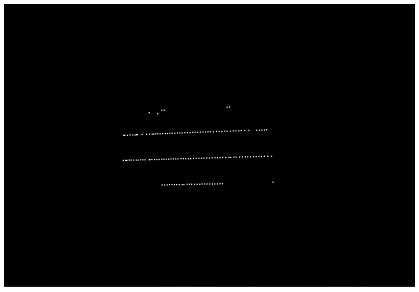

Fig. 1.5 16-channel LiDAR scans reflecting on a single side of a vehicle.

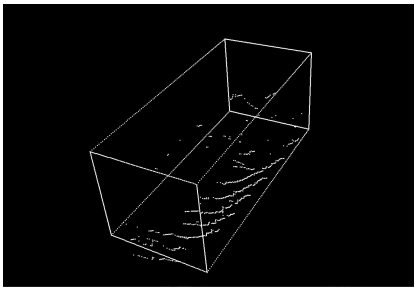

Fig. 1.6 Bounding Box generated using LiDAR based classification.

1.2.4 Detection and Pose Estimation of Vehicles

For obstacle detection and avoidance, it is essential to know the size, shape, and ultimately the centroid and classification of a detected object with a high degree of reliability. While LiDAR data provides estimates with centimeter level accuracy, it can be challenging to estimate the pose of an object out of a single point cloud. For vehicles, laser scans may reflect off a single side of the vehicles making it challenging to estimate their true width, as shown in Figure 1.5. Additionally, the LiDAR vertical field of view and resolution can deteriorate significantly for objects that are further away depending on the number of LiDAR channels.

In the ideal case in which LiDAR data is provided for two adjacent sides of a vehicle, we can extract an L-shaped point cloud when it is projected onto the XY plane. Thus we can estimate the pose based on this information. Bounding box and orientation fitting methods [14] have been explored, but in this section, we describe an alternative RANSAC based line fitting method that can also be applied for this application.

Based on the assumption that most extracted points define two nearly-orthogonal intersecting planes, we first project our data on a plane and find the first best-fit orthogonal line using a RANSAC approach and remove all the points that are relatively close to this line. The second line is estimated with the remaining points while enforcing the constraint that it should be perpendicular to the first line. Based on the two lines and prior knowledge of vehicle sizes, we approximate and fit a bounding box for the corresponding point cloud, as shown in Figure 1.6.

In realistic scenarios, these ideal cases are seldom encountered. During the tests performed, vehicles that are located along the XY axis, with respect to the LiDAR coordinate frame, project a single side and do not provide enough information to estimate width. Integration of relative position, lane information and even tracking information is needed for a more robust vehicle classification and tracking system. On the other hand, for obstacle detection and avoidance, it is sufficient to cluster points together to represent an unlabeled obstacle.

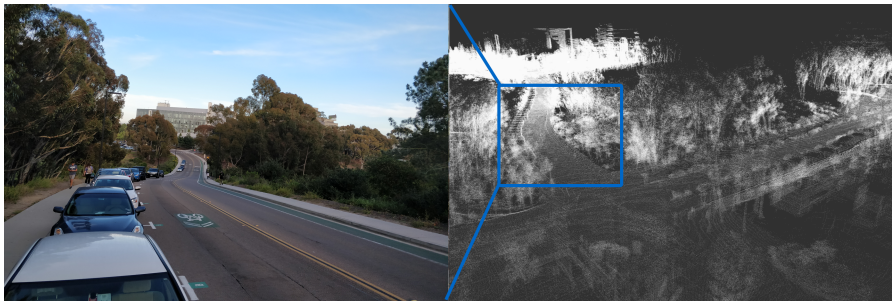

Fig. 1.7 UC San Diego: Voigt Drive hill shown on the left with the associated point cloud map shown on the right.

1.2.5 Vehicle Control and Path Tracking

The trajectory tracking algorithm used in Autoware is the geometric Pure Pursuit algorithm, which considers the upcoming waypoint as the desired pose and computes the required linear and angular velocities based on a geometric curve fit. Keeping in mind the environment and the low speeds (a maximum of 25 mph) at which we operate our system inside the university campus, once the reference trajectory and its curvature is considered, we employ a PID strategy for the longitudinal control and a bicycle model of the car for the steering control with an angular velocity that is directly proportional to the current speed of the vehicle. Given that the acceleration and braking capabilities of the car are different due to inherent mechanical constraints, it is also necessary to have different PID controllers for acceleration and braking. Extensive testing was done in this regard to achieve the appropriate constants for acceleration and braking that suited varied road inclines present on campus as shown in Figure 1.7.

When coupled with a purely geometric path tracking algorithm, the cornering abilities of the driving are questionable at higher speeds. Cases where the vehicle shot off the path while driving at relatively higher speeds around sharp corners were experienced, which led us to explore other strategies to help tackle this issue. In view of that, the kinematic model for path following [11] was studied and researched upon for the current application. However, after careful analysis of the advantages of the pure kinematic model, it was found to be comparable at lower speeds to that of our current version and without any additional benefits.

As part of an ongoing research, we seek inspiration from one of the major areas of current research in autonomous driving, Lane Keeping Assists Systems (LKAS). A Vision-Based Lane Assist System using deep learning models like DeepLanes [5] and SCNN [13] to estimate the vehicles position and heading in the lane, and thereby obtain the cross track error and desired orientation of the vehicle, is being explored. On the desired trajectory obtained, a Model Predictive Control (MPC) approach is followed as the control strategy. With the reference trajectory coming in

from vision-based lane detection, the MPC controller seeks to minimize the cross-track error and the errors in heading direction and velocity of the car. (1) represents the cost the MPC controller is trying to minimize. The constraints that are derived from the kinematic model and those that govern the optimization are shown in (2).

$$J = \Big(\sum_{t=0}^{t=N-2} K_1\, e_{cte,t}^2 + K_2\, e_{\psi,t}^2 + K_3\, (v_t - v_{ref})^2 \Big)$$
$$+ \Big(\sum_{t=0}^{t=N-2} K_4\, a_t^2 + K_5\, \delta_t^2 + K_6\, (a_t - a_{t-1})^2 + K_7\, (\delta_t - \delta_{t-1})^2 \Big) \tag{1}$$

$$\begin{aligned} &x_t + v_t \cos(\psi_t)\, dt - x_{t+1} = 0 && y_t + v_t \sin(\psi_t)\, dt - y_{t+1} = 0 \\ &v_t + a_t dt - v_{t+1} = 0 && \psi_t + v_t \tfrac{\delta_t}{L_f} dt - \psi_{t+1} = 0 \\ &f(x_t) - y_t + v_t \sin(e_{\psi,t}) - e_{cte,t} = 0 && \psi_t - \psi_{ref} + v_t \tfrac{\delta_t}{L_f} dt - e_{\psi,\, t+1} = 0 \end{aligned} \tag{2}$$

x_t, y_t, ψ_t : *pose at time t*
v_t : *velocity at time t*
a_t, δ_t : *acceleration and steer at time t*
$e_{cte,t}$: *cross track error at time t*
$e_{\psi,t}$: *error in heading at time t*
$f(x_t)$: *reference trajectory*
v_{ref} : *reference velocity*
ψ_{ref} : *desired heading*
$K_{1:5}$: *weighting constants*
L_f : *wheelbase of the car*

1.2.6 AVL Logger

As of January 2019, the California Department of Motor Vehicles has issued 62 Autonomous Vehicle Testing permits to large vehicle manufacturer corporations and start-ups to perform field tests with a safety driver. As part of the California Autonomous Vehicle Testing Regulations, every manufacturer is required to submit an annual report with a summary of system disengagements. Although most publicly available reports include a summary with the number of disengagements over a period of time, there are many important aspects that are missing. Some of these aspects that are not evaluated include Mean Time Between Interventions (MTBI), Mean Distance Between Interventions (MDBI), comparisons between autonomous vs. manual driving, AV impact on power consumption, and long-term implications such as maintenance and overall cost of ownership. These metrics require a more comprehensive methodology for benchmarking a self-driving vehicle and the implications of long-term autonomy.

As part of an ongoing research effort, AVL has developed a system for standardizing these metrics. The logging system shown in Figure 1.8. was designed to log various signals for performance and disengagement analysis in a non-intrusive isolated environment. The logging system is based on Flask and the RESTful API on a Raspberry Pi. The signals collected from the autonomous vehicle computing platforms are transferred through HTTP(S) protocol via ethernet and stored in SQLite databases for future offline analysis (Figure 1.9). This vehicle API is a lightweight

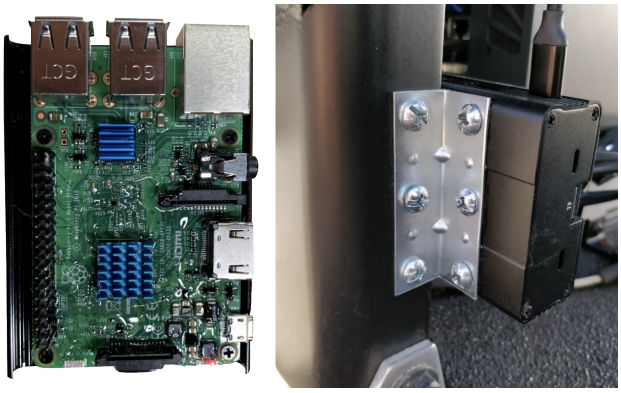

Fig. 1.8 Raspberry Pi Logging Module mounted on a development platform

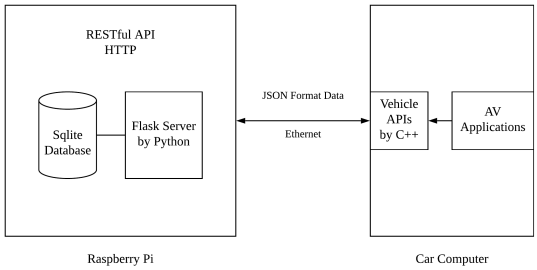

Fig. 1.9 Logging System Architecture

software package (ROS node) which serializes all of the logging information and encapsulates it using JSON objects.

This external logging mechanism is designed to be used continuously without interfering with the main system and can be beneficial for third-party verification by public-key encryption. Some of the metrics defined for the vehicles being tested at UC San Diego are listed on Table 1.1.

1.3 Experimental Results

Live tests and data collection are actively performed at UC San Diego for development under different terrain conditions. As seen in Figure 1.10, the topological conditions and geometry of the university provide testing grounds in terms of evaluating the robustness of mapping and localization, perception and tracking, motion planning, and logic.

Trigger	Metric	Type
Intervention	Mean Distance Between Intervention (MDBI)	Event Driven
Intervention	Mean Time Between Intervention (MTBI)	Event Driven
Energy	Miles per Gallon (MPG) or Charge (mAh)	Continuous
Maintenance Cost	Brakes and Tire Wear	Event Driven
Up-time	Time Elapsed Per Trip	Continuous
Control	Speed, Acceleration, Steer Angle	Continuous
Location	GPS / Map Coordinates	Continuous

Table 1.1 Metrics defined for bench marking and evaluation of autonomous vehicles. Typical values for MDBI and MTBI during initial autonomous tests were 0.53 km/intervention and 2.6 min/intervention, respectively. An ongoing study is being conducted to analyze long term autonomy using these as primary metrics.

1.3.1 Localization and Mapping

Extensive localization and mapping tests have been performed at UC San Diego using Normal Distributions Transform (NDT) [1]. The Point Cloud Library (PCL) implementation provides a library for mapping and localization and involves a three-step process: data collection, mapping, and post-processing. In the data collection step, a development platform with a sixteen channel LiDAR fixed on the roof of the vehicle was used to collect data by driving along multiple parts of campus. Depending on the size of the data, mapping was performed on a CPU or a GPU. The use of a graphics card can significantly accelerate the process of generating point cloud maps; however, as maps grow large in size, graphics cards begin to run out of internal memory causing the mapping algorithms to crash. This process was tested on a variety of CPUs and GPUs including the Intel Xeon E3-1275, Intel i7-6700, Intel i9-7900, Nvidia Quadro M1000M, Nvidia GTx 1080Ti, and Nvidia Titan Xp.

Figure 1.10 is an example of a map that is actively used to perform tests on campus. The generation of this map required the use of CPU with a computation time of approximately 28 hours; the total distance along the trajectory is 3.9 miles. In the post-processing step, the map was down sampled using a Voxel Grid Filter approach with $30cm^3$ 3D grids. [9].

The size of the original map generated was 2.8GB and was reduced to 530MB after downsampling. During exhaustive testing, it was determined that as point cloud maps exceed 500GB in size, localization and system performance begins to be impacted significantly and in the worst case (1GB and above) cause the algorithm to crash or lag long enough to lose the localization prediction. This poses a major challenge in scalability for self-driving stacks that are highly dependent on dense point clouds and LiDAR based mapping and localization.

Given that NDT employs a stochastic approach, it is robust to certain discrepancies between the original map and changes to the environment but at the same time becomes more complicated to estimate future system failures. Although immediate system failures are detectable by measuring the NDT matching error score, addi-

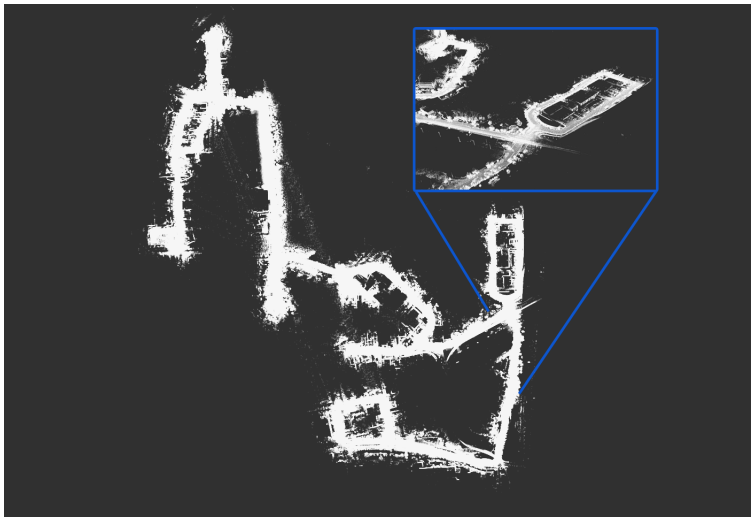

Fig. 1.10 3D Point cloud map corresponding to mail delivery routes at UC San Diego. The distance of the trajectory is approximately 3.9 miles.

tional sensor data fusion is essential for addressing potential risks that may originate from unexpected behavior - ultimately improving robustness.

1.3.2 Motion Planning and Logic

The California Department of Motor Vehicles defines an intersection to be a place in which one roadway meets another roadway including cross streets, side streets, alleys, freeway entrances and any other location where roads join each other that can include unmarked crosswalks [3]. The anticipated mail delivery route at UC San Diego contains 22 crosswalks, some of which are protected by stop signs, and others that require drivers judgement to slow down or stop.

Our crosswalk state implementation is an extension of the OpenPlanner behavior state machine, and contains two additional states: crosswalk approach and crosswalk stop. In the crosswalk approach state, the ego vehicle will reduce its velocity as it becomes within a certain distance the crosswalk. The vehicle will maintain a reduced velocity until it passes the crosswalk and return to its target speed. However, if pedestrians appear on the crosswalk, the vehicle will enter the crosswalk stop state, stop before the crosswalk, and wait for all pedestrians to clear before moving again. A diagram detailing the crosswalk states is shown in Figure 1.11.

Addition of the crosswalk class entailed modifications in the OpenPlanner road network class, along with all its supporting functions. However, we note that challenges were encountered during implementation of behavior states in Autoware,

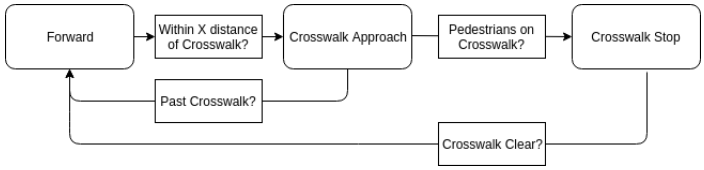

Fig. 1.11 Introduced Crosswalk State in Finite State Machine.

primarily due to the proprietary AISAN vector map format. One issue encountered was that crosswalks could only be detected when driving on a single lane, but not along the opposite direction. This stemmed from the fact that each encoded crosswalk feature was linked with a single waypoint, which in turn is part of a single lane. In order to link to all the lanes that the crosswalk intersects, we linked the crosswalk to all lanes that were within a certain radius.

By a similar approach, the foundation for intersection logic is based on the location of all of the stoplines at an intersection; these are encoded in the vector map to determine if a vehicle is waiting at the stop sign or not.

Once the ego vehicle comes to a stop and enters the three second rule required by law, the ego vehicle enters a wait state until an intersection-clear signal is transmitted that allows it to resume its intended trajectory as shown in Figure 1.12. To determine if the ego vehicle is clear to cross an intersection, a vehicle-to-stop sign association must be performed to determine the order in which the other vehicles arrived, if any. Presently, the intersection module is under constant development as perception and tracking is completed. Using a distance metric and camera information, one approach being explored involves a simple plane-to-plane mapping using homographies [12] and real-time vehicle classification data obtained from our four front cameras with a horizontal field of view of approximately 180°.

The current crosswalk and intersection state implementations have been verified via simulation with the modified version of OpenPlanner. The integration of pedestrian detection and intersection logic is ongoing and is expected to be completed in the near future.

1.4 Conclusion

Numerous lessons were learned during the process of enabling last mile transportation where both pedestrian and automotive traffic are present. The first and foremost concern in last mile transportation is safety, which hinges on accurate vehicle control, LiDAR detection, localization and trained safety drivers.

We surveyed several control algorithms, including PID kinematic control and regular kinematic control. Despite efforts to fine tune the models, both displayed characteristics of overshooting lane boundaries and other hazardous behavior when tested with our development platforms at high speeds. To guarantee safe behavior

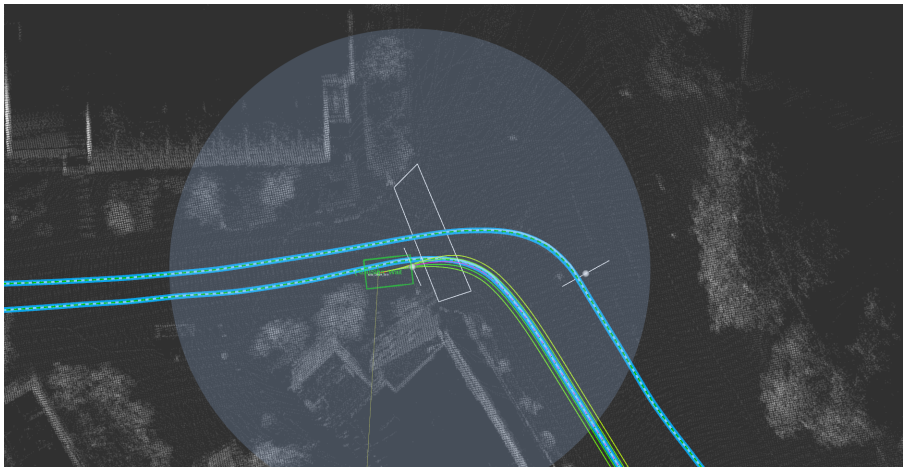

Fig. 1.12 Simulated ego vehicle reaches a stop line and awaits for intersection-clear signal. Center stop lines are shown by white markers and are used for finding stop lines within a 30m search radius (shown in blue) from an intersection.

of autonomous vehicles, an MPC approach, in combination with vision based lane detection can potentially minimize the caveats we faced with simpler models. We look forward to evaluating this model with our platforms in the future.

While evaluating LiDAR vehicle detection, the assumption of flat ground is insufficient for the hilly terrains encountered at UC San Diego. Since the ground could take up varying portions of the LiDAR points, a ground removal step is necessary for reducing false positives and increasing the accuracy of vehicle detections in a scene. One alternative being explored involves multi-camera and LiDAR fusion with deep neural networks using YOLO3 [7].

LiDAR based localization and mapping is currently a critical component of autonomous navigation in our implementation of last mile transportation. While the implementation of this technology is feasible, as determined in the course of this study, there are caveats that follow. For micro-transit and smaller scale transportation systems specifically, this technology proposes robust and adequate results given that the maps and vector maps are constantly maintained and updated. On the other hand, as the scale of the environment drastically increases in size and complexity (i.e small cities or towns), performance is hindered and can result in detrimental errors without additional safeguards in place and extensions that incorporate the notion of context switching between smaller maps. This is an area of research that needs to be addressed for larger scale systems. To address this challenge, AVL is developing a software stack that eliminates the notion of dense point clouds and LiDAR based localization by introducing a planner that performs under a nominal set of high level instructions. This approach is intended to handle planning and localization in a similar fashion that human drivers operate with the benefits of 3D situation awareness and dynamic obstacle avoidance.

During this process, we have gained a deeper understanding of open-source state of the art architectures, strengths, and shortcomings for last mile transportation. With our continued focus and efforts to enable autonomous vehicles at UC San Diego, we expect to create a safer environment and more efficient campus transportation.

Acknowledgements We acknowledge the support provided by UC San Diego Logistics Unit, Fleet Services, Police Department, and Mail Delivery Center. We would like to thank Dr. Henrik I. Christensen and Dr. Todd Hylton for making this partnership and project possible, as well Shengye Wang, Dominique Meyer, Eric Lo, Sayan Mondal, Shawn Winston, Francis Joseph, and Ploy Temiyasathit for contributing on different aspects of the project.

References

1. Almqvist, H, Magnusson, M, Kucner, TP, Lilienthal, AJ. (2018) Learning to detect misaligned point clouds. J Field Robotics. 2018; 35: 662 677
2. Autoware: Open-source software for urban autonomous driving. `https://github.com/CPFL/Autoware/` Accessed: Accessed 10 Apr 2019
3. California Department of Motor Vehicles (2019) California Driver Handbook - Laws and Rules of the Road. `https://www.dmv.ca.gov/portal/dmv/detail/pubs/hdbk/right_of_way/`. Accessed 10 Apr 2019
4. Darweesh H, Takeuchi E, Takeda K, Ninomiya Y, Sijiwo A, Morales Y, Akai N, Tomizawa T, Kato S (2017) Open Source Integrated Planner for Autonomous Navigation in Highly Dynamic Environments. Journal of Robotics and Mechatronics 29(4):668-684. doi: 10.20965/jrm.2017.p0668
5. Gurghian A, Koduri T., Bailur SV, Carey KJ, Murali VN (2016). DeepLanes: End-To-End Lane Position Estimation Using Deep Neural Networks. 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 38-45 June 2016
6. Quigley M et al (2009) ROS: an open-source Robot Operating System. Paper presented at ICRA Workshop on Open Source Software, Kobe, 12-17 May 2009
7. Redmon J, Farhadi A (2018) YOLOv3: An Incremental Improvement. CoRR
8. Romanoni A, Matteucci M (2016) Robust moving objects detection in lidar data exploiting visual cues. Paper presented at IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, 9-14 October 2016
9. Rusu RB, Cousins S (2018) 3D is here: Point cloud library (PCL). Paper presented at IEEE International Conference on Robotics and Automation, Shanghai, 9-13 May 2011
10. Shengye W, Christensen HI (2018) TritonBot: First Lessons Learned from Deployment of A Long-term Autonomy Tour Guide Robot. Paper presented at 2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Nanjing, August 2018
11. Snider JM (2009) Automatic Steering Methods for Autonomous Automobile Path Tracking. Dissertation, Carnegie Mellon University
12. Torr PHS, Zisserman A (1999) Feature Based Methods for Structure and Motion Estimation. In: Proceedings of the International Workshop on Vision Algorithms: Theory and Practice, London, 21-22 September 1999
13. Xingang P et al (2018) Spatial as deep: Spatial cnn for traffic scene understanding. In: Thirty-Second Association for the Advancement of Artificial Intelligence Conference, New Orleans, 2-7 February 2018
14. Zhang X, Xu W, Dong C, Dolan JM (2017) Efficient L-Shape Fitting for Vehicle Detection Using Laser Scanners. Paper presented at IEEE Intelligent Vehicles Symposium, Redondo Beach, 11-14 June 2017