Autonomous Intelligent Systems

ORIGINAL ARTICLE Open Access

Autonomous vehicles for micro-mobility

Henrik Christensen, David Paz*, Hengyuan Zhang, Dominique Meyer, Hao Xiang, Yunhai Han, Yuhan Liu, Andrew Liang, Zheng Zhong and Shiqi Tang

Abstract

Autonomous vehicles have been envisioned for more than 100 years. One of the first suggestions was a front cover of Scientific America back in 1916. Today, it is possible to get cars that drive autonomously for extended distances. We are also starting to see micro-mobility solutions, such as the Nuro vehicles for pizza delivery. Building autonomous cars that can operate in urban environments with a diverse set of road-users is far from trivial. Early 2018 the Contextual Robotics Institute at UC San Diego launched an effort to build a full stack autonomous vehicle for micro-mobility. The motivations were diverse: i) development of a system for operation in an environment with many pedestrians, ii) design of a system that does not rely on dense maps (or HD-maps as they are sometimes named), iii) design strategies to build truly robust systems, and iv) a framework to educate next-generation engineers. In this paper, we present the research effort of design, prototyping, and evaluation of such a vehicle. From the evaluation, several research directions are explored to account for shortcomings. Lessons and issues for future work are additionally drawn from this work.

Keywords: Autonomous vehicles, Intelligent systems, Micro-mobility

1 Introduction

Design of autonomous vehicles is not a new effort. One of the earliest efforts was led by Dickmanns at the Universitaet der Bundeswehr in Munich. Dickmanns developed an autonomous navigation stack based on computer vision techniques using the VaMoRs vehicle; early publications are from 1986 [1] and the project is summarized in [2]. About the same time, the US launched its Autonomous Land Vehicle (ALV) program as part of the Strategic Computing Initiative [3]. By 1995, Carnegie Mellon University completed a 2,850 mile trip from Pittsburgh to San Diego; 98.2% of the trip was driven autonomously[1]. Subsequently, programs such as the European Prometheus (1987-1995) and the DARPA Grand Challenge programs (2004-2007) generated a lot of attention around the technology, which in turn has provided key people to launch efforts at Waymo [4], Toyota [5], Zoox [6], Aurora [7], etc. Although tremendous improvements and state-of-the-art developments have been introduced in recent years, in many cases the driving is on major roads and freeways with limited interaction with other road-users beyond cars. Today, autonomous navigation at scale and in urban scenarios largely remains an open challenge.

To quantify the challenges associated with last-mile autonomous navigation and micromobility, we decided to deploy a set of vehicles for urban operation. Rather than trying to build our own vehicles, our team acquired off-the-shelf vehicles and had them retrofitted for computer control. Our focus is on the sensor suite, sensor calibration and fusion, perception with minimal maps, robust path planning and execution on a standard platform. We utilize two GEM e6 golf carts. Given that our campus already has 25+ GEM vehicles, it is a known entity and easy to service for personnel in the vehicle pool. The cars were retrofitted for computer operation using a CAN bus solution and standard Linux software. The retrofit was performed by AutonomousStuff. For initial experiments, both the Baidu Apollo system [8] and the Tier IV Autoware system [9] were considered. The Autoware software stack is based on the ROS-1 and Ubuntu Linux. Most of the team had ROS experience and as such it was a path of the least development resistance.

[1] http://www.cs.cmu.edu/\protect\unhbox\voidb@x\penalty\@M\tjochem/nhaa/Journal.html

*Correspondence: dpazruiz@ucsd.edu
University of California, San Diego, La Jolla, California 92093, USA

(a) (b)

Fig. 1 The GEM e6 vehicles and the associated sensor layout

We present a comprehensive description and analysis of the design, implementation, and evaluation of vehicles for urban operation. We will provide a more detailed view of the initial design in Section 2. The main lessons / shortcomings from the deployment are analyzed in Section 3 and various approaches to address these challenges are presented in Section 4. We outline a number of research challenges for the future in Section 5.

2 Initial build

In this section, the initial architecture designs are introduced given the requirements from Section 1. These include a description of the development platforms, software containerization, sensing methodologies, as well as the detection, decision making, and control strategies. The section concludes with additional details on closed-loop testing and system verification to ensure real-time capabilities.

2.1 Systems, sensors, and software tools

The development platforms consist of two six-seat GEM e6 golf carts (Fig. 1(a)); these were retrofitted with drive-by-wire (DBW) systems, computing modules, logging devices, and multiple suites of ranging and vision sensors. The DBW systems provide each vehicle platform with two-way communication to execute control commands while simultaneously providing actuator and motor status reports for close-loop control. The interfaces include steering wheel angle control, angular velocity, braking and acceleration input, and provide status reports for speed, DBW enable/disable signal, as well as steering, accelerator, and brake states. The instructions delegated to the DBW system are generated after considering perception, decision-making, and high-level control commands which are performed from the main onboard computer: each primary system consists of an Intel Xeon E3-1275 Processor with 32 GB RAM and an NVIDIA GTX 1080 Ti GPU. Additional system, logger, and sensor details can be found in [10].

2.1.1 Sensors

Each vehicle was retrofitted with multiple Mako G319 cameras, Velodyne VLP-16 LiDARs, and Bosch Ultrasonic sensors. An overview of each sensor configuration is shown in Fig. 1(b). Intrinsic calibration was performed manually for each camera using a standard checkerboard [11] to determine the projection characteristics between the camera and image frames. On the other hand, extrinsic calibration was performed between each camera-LiDAR pair by manually identifying correspondences in the camera and LiDAR frames. Camera-LiDAR calibration is an essential process to leverage projective geometry techniques. Additional calibration tools were later developed as described in Section 4.1 to partially automate the extensive calibration process.

2.1.2 Logging

To facilitate offline data post-processing and analysis, a data logger was designed for capturing status reports from each of the vehicles' actuators, system enable/disable signals, nominal GPS and ego-vehicle pose updates, and control inputs. Given that raw sensor data can require high bandwidth and storage requirements, sensor data is separately recorded using the ROSBAG file format [12]. The Linux Epoch timestamps are utilized to provide a reference between the logger and bag file data stored. This logging device is based on Flask and a RESTful API that was designed by serializing the reported vehicle signals over an Ethernet connection; these signals are then recorded in an SQLite database within a Raspberry Pi module. The modular design provides a plug-and-play approach for data logging that is automatically initiated.

2.1.3 Software tools and containerization

An important software strategy incorporated into the design of the system involves software containerization using Docker. This strategy is motivated by initial challenges encountered when integrating software packages and drivers within a single computing platform. Various open-source packages and libraries initially utilized in the design of the algorithms often required dependencies that were incompatible with one another. To address incompatibility constraints, software packages and related tools were separated into multiple containers with various image definitions, as shown in Fig. 2. An additional benefit of software containerization involves cross-platform compatibility, such as different operating systems; this enables flexible software development across various types of platforms and was essential during development of our software.

2.2 The ROS / autoware framework

Designing complex software stacks for autonomous driving requires design considerations for robust communication across various hardware, software, sensor, and actuator components. Given the complexity of a full-scale vehicle system, abstraction and modularity become essential design considerations during development and verification. To facilitate this process, our approach for the design of our system leverages the Robot Operating System (ROS) [12] and Autoware [9]. ROS is an open-source suite of libraries and tools that enable the design of various robotic applications and provides communication across various types of software, sensor, and actuator modules. A particular ROS based framework that was utilized as part of the foundation of our work is Autoware. The Autoware package suite includes a collection of open-source modules that provide certain features for localization, mapping, perception, and planning.

In this section, we cover the hierarchical design aspects of our software stack that extend from Autoware and ROS. A high-level design diagram of the system is shown in Fig. 3.

2.2.1 Perception

Perception is a key component within the hierarchical architecture introduced; it provides the states of potential obstacles to be able to effectively follow vehicles in front of the ego-vehicle and prevent collisions in the

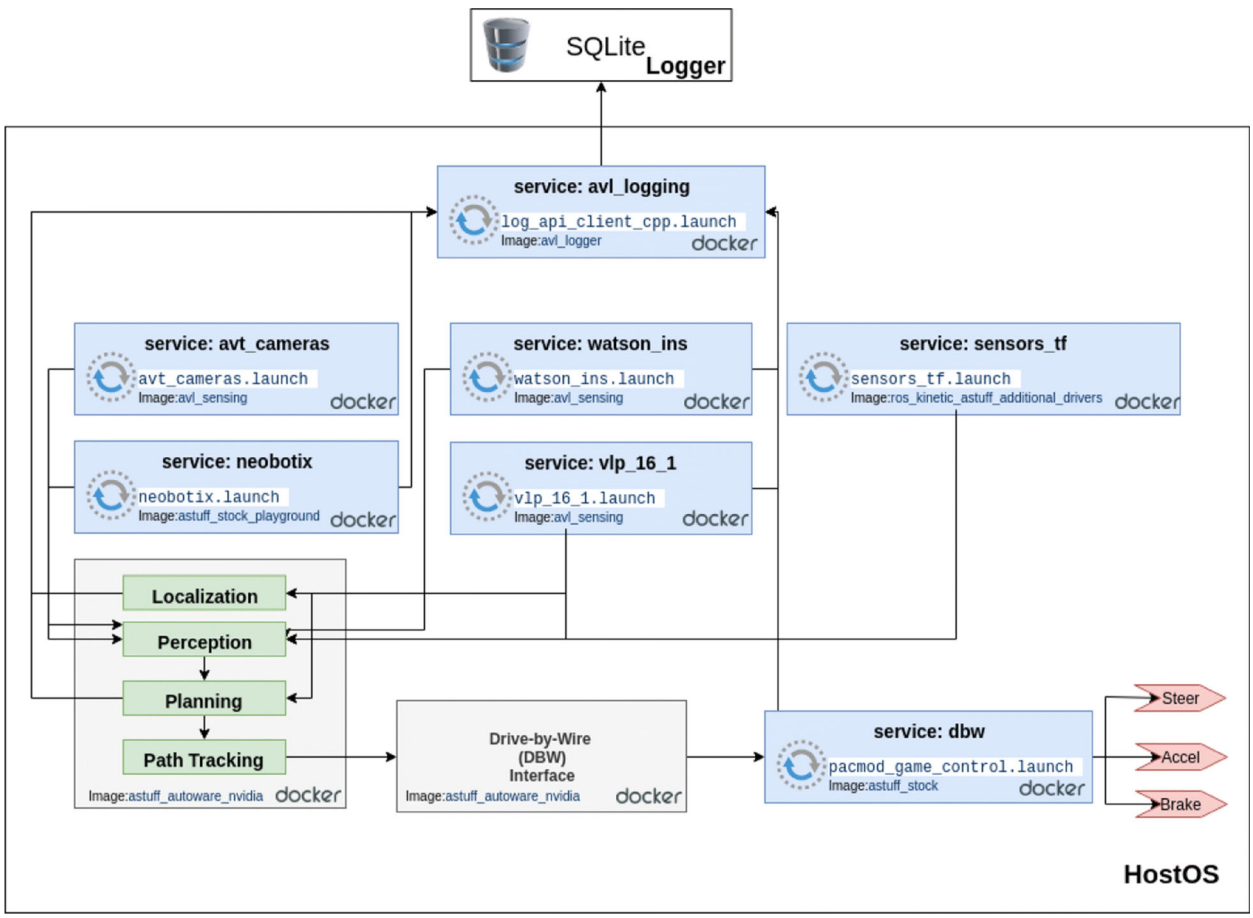


Fig. 2 Initial system organization using containerized versions of Autoware and various ROS packages

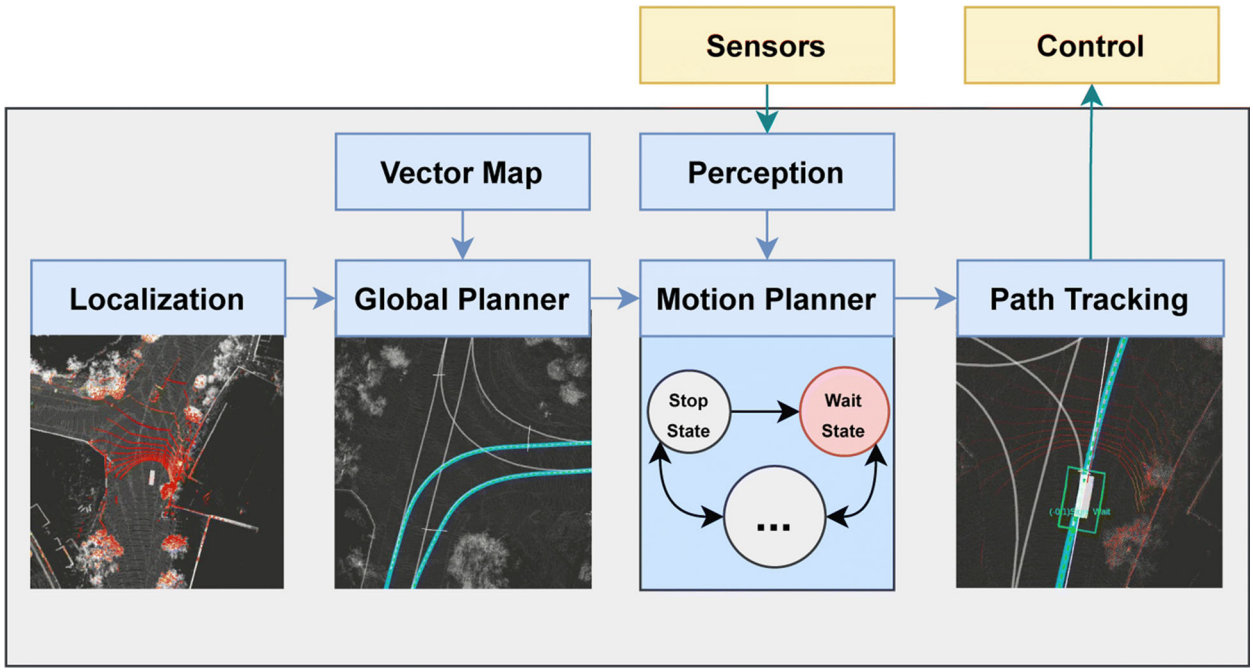


Fig. 3 Hierarchical autonomous driving stack comprised of localization, perception, global planner, motion planner, and control modules

motion planning phase. Our perception module includes LiDAR based detection and clustering techniques, as well as camera-LiDAR fusion techniques to provide contextual information from image representations.

LiDAR Detection and Tracking

Recent developments in learning-based 3D object detection leverage LiDAR data to regress 3D bounding boxes for vehicles and pedestrians [13]. Although robust methodologies have been deployed with real-time characteristics, most of these depend on LiDAR sensors with higher spatial resolution such as 32-channel or 64-channel LiDARs. Instead, our approach uses traditional object clustering for object detection with a 16-channel Velodyne LiDAR that provides 360° horizontal field of view and ±15° vertical field of view.

An important consideration while designing this LiDAR based clustering approach for detection is that relevant obstacle information may be combined with objects with minimal interest, such as road surfaces. Therefore, without removing road surface data, it becomes non-trivial to distinguish obstacles during planning. The approach utilized in this work for segmenting obstacles from road surfaces involves ground removal. Autoware leverages this approach by identifying the LiDAR's pitch angle, defining a plane normal to the ground, and eliminating points that fall below a threshold. However, this approach generates a high-rate of false detections for regions that are non-planar, such as steep hills. Therefore, rather than leveraging a single plane fitting methodology, our approach uses multi-plane definitions to separate ground points from potential obstacles using the Random Sample Consensus (RANSAC) algorithm [14]. Figure 4 provides a visualization of the approach, and a visualization of the clustering approach is shown in Fig. 5—where the turquoise point cloud corresponds to LiDAR points above the ground and the yellow scans correspond to the ground plane. For obstacle detection, the LiDAR data that corresponds to potential obstacles is then clustered based on the relative distance and nearest neighbors to perform the association and extract individual objects.

After extracting individual clusters, each object is associated across time to reason about an object's states that depend on temporal information; this includes velocity and acceleration. This association is performed using a Kalman Filter tracker based on each object's cluster centroid and by keeping track of timestamp information over time. While this approach works well for tracking smaller shape objects, variations in performance are observed for large objects such as large vehicles depending on the relative orientation and shape characteristics given that the centroid of each cluster can change. Furthermore, the performance of LiDAR based detection methods can depend on the spatial resolution of the sensor and maximum perception range. Methods for addressing these shortcomings are introduced in Section 4.2.

Camera-LiDAR Fusion

While LiDAR sensors can provide range information with high accuracy for various objects in the scene,

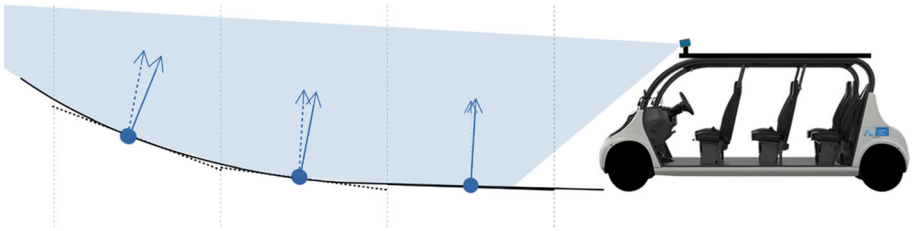


Fig. 4 Recursive ground plane filtering from close to far for curved roads. The solid arrows reflects the normal direction of the planes and dash arrows reflects the initial guess normal

the decision-making process needed to perform complex maneuvers often requires contextual information such as object classification and texture information. This is information that can be readily accessible from image and learning based methods but becomes non-trivial to extract from raw LiDAR data. Hence, to further enhance our LiDAR based detection, we leverage a camera-LiDAR fusion approach.

To identify representative objects and features in a camera frame from the LiDAR perspective and vice versa, a calibration process is performed. Calibration seeks to identify the geometric relationship between two or more frames of reference. This extrinsic relationship ($\mathbf{T} \in \mathrm{SE}(3)$) can be captured by matching features in the frames of interest and solving the Perspective-n-Point problem (PnP) [15]. The generated camera-LiDAR relationships can be visualized in Fig. 6, where the relative transformation between the LiDAR L and a camera C_i corresponds to ${}_{C_i}\mathbf{T}_L$.

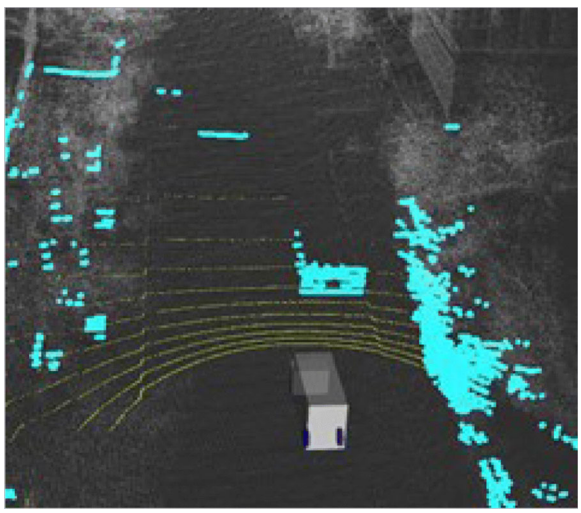


Fig. 5 Clusters of points are separated (in turquoise) from the ground points filtered (in yellow)

To match features between each camera-LiDAR pair, a visualization toolkit from Autoware was utilized to manually identify features in the image frame and LiDAR frames simultaneously.

With these extrinsic properties, a projective geometry approach can be utilized to project LiDAR clusters generated by the detection module into image frame. The relationship between a LiDAR cluster $\mathbf{X}_j$ and its corresponding representation in image i in terms of pixels is given by $\mathbf{x}_j = \mathbf{K}_i \cdot {}_{C_i}\mathbf{T}_L \cdot \mathbf{X}_j$, where $\mathbf{K}_i$ corresponds to camera i's intrinsic parameters that describe projection characteristics. Once the corresponding pixel coordinates for each cluster are estimated, learning based multi-object detectors such as YOLOv3 [16] can be used to apply classification labels as shown in Fig. 7.

The intrinsic parameters for each camera are manually identified using a standard checkerboard calibration toolkit [11]. Given the extensive calibration process required by our approach, techniques for partially automating this process are explored in Section 4.1.

In the next section, ego-vehicle localization (${}_B\mathbf{T}_M$) is leveraged to unify static map elements and perception objects within the same reference frame. Given that the ego-vehicle's pose is continuously updating, ${}_B\mathbf{T}_M$ is a moving frame. Finally, the map frame $\mathbf{M}$ is used to represent obstacles and perform avoidance maneuvers during motion planning.

2.2.2 Mapping and localization

Simultaneous localization and mapping (SLAM) is a topic that has been studied extensively in the robotics community using vision and LiDAR methods [17, 18]. This enables route and motion planning strategies that require the agent's pose estimate to execute the next action, making it a necessary module in an autonomous stack.

Our work utilizes the concepts of Three-Dimensional Normal Distributions Transform (NDT) [19] for mapping and localization. In the mapping process, we perform a data collection process at the UC San Diego campus by driving along the routes of interest and recording LiDAR data. This data is then used to generate dense

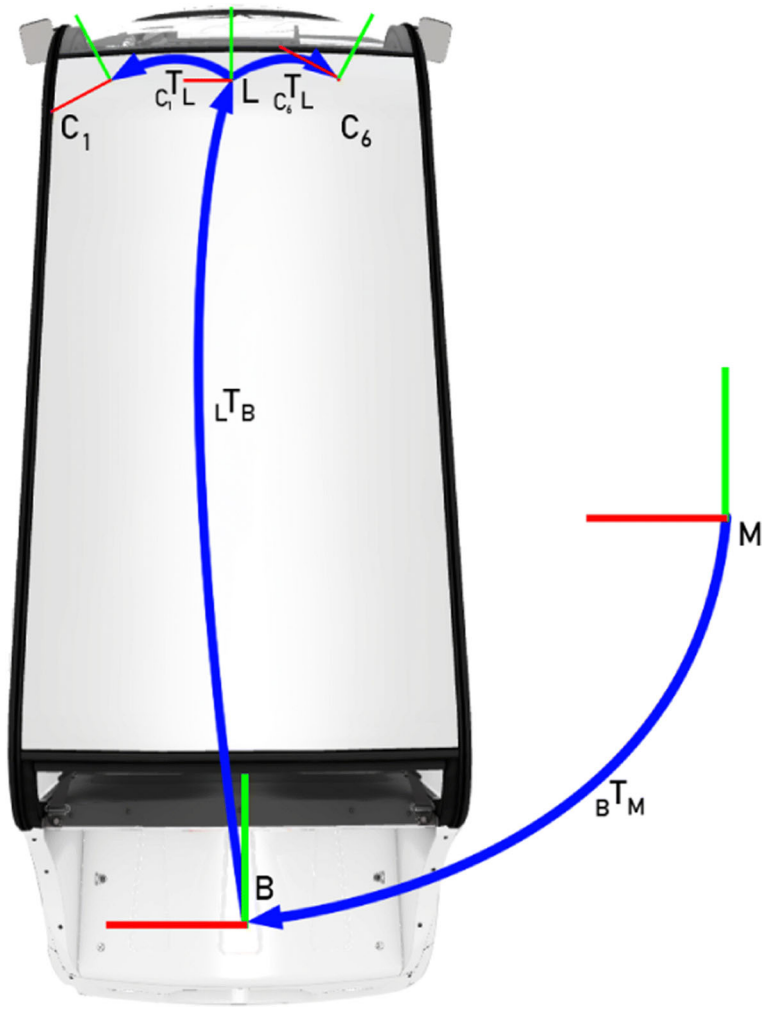
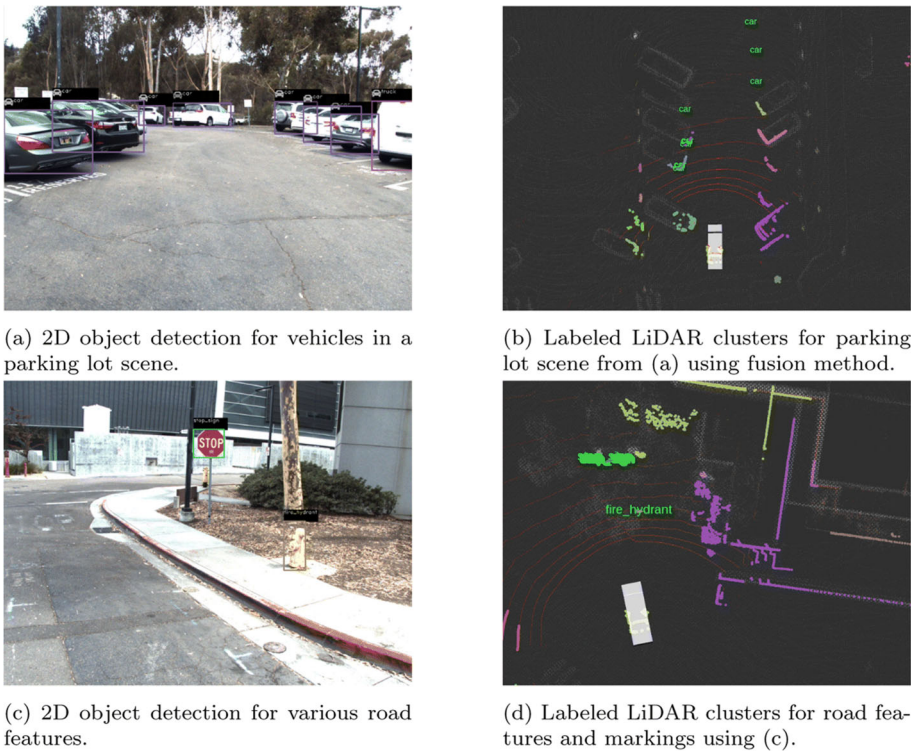


Fig. 6 Relative transformation from the map frame to the vehicle frame (baselink) ${}_{B}\mathbf{T}_{M}$, from the vehicle frame to the LiDAR frame ${}_{L}\mathbf{T}_{B}$, and from the LiDAR frame to the camera *i* frame ${}_{C_i}\mathbf{T}_{L}$

point cloud maps using the Point Cloud Library (PCL) implementation [20] for NDT. A Voxel 3D Grid filter is additionally applied to downsample the generated point cloud. A visualization of the downsampled 3D point cloud generated from the campus is shown in Fig. 8(a); this map corresponds to approximately 3.9 miles of drivable road segments and requires 485MB of storage compared to 2.7GB before downsampling. With this map, NDT is also applied to localize the ego-vehicle with respect to the map using LiDAR based scan matching, this can be visualized in Fig. 8(b). This effectively provides continuous ego-vehicle pose estimates with respect to the map's origin (${}_{B}\mathbf{T}_{M}$) as discussed in the previous section (Fig. 6).

While understanding the pose of the vehicle over time is necessary for point-to-point navigation, additional contextual information is needed for estimating short term control actions. This includes identifying lane markings, traffic signals and centerlines for various scenarios including intersection navigation and lane following. To facilitate this process, our team incorporated additional map features with respect to the point cloud map. These annotations are often referred to as Vector or High-Definition (HD) maps. Similar maps have been released in various open-source datasets [21, 22] over the years with extensive manual labels such as parking areas, exact 3D traffic signal locations, lane types, and sidewalks. In comparison, we simplify maps and only annotate centerlines, stop lines, crosswalks, and speed limits as shown in Fig. 9. Despite reducing the number of features annotated, the manual annotation and verification process for the UCSD campus took a week. In Section 3.2.3, we discuss the limitations with this approach and introduce alternative methodologies in Section 4.3.



(a) 2D object detection for vehicles in a parking lot scene.

(b) Labeled LiDAR clusters for parking lot scene from (a) using fusion method.

(c) 2D object detection for various road features.

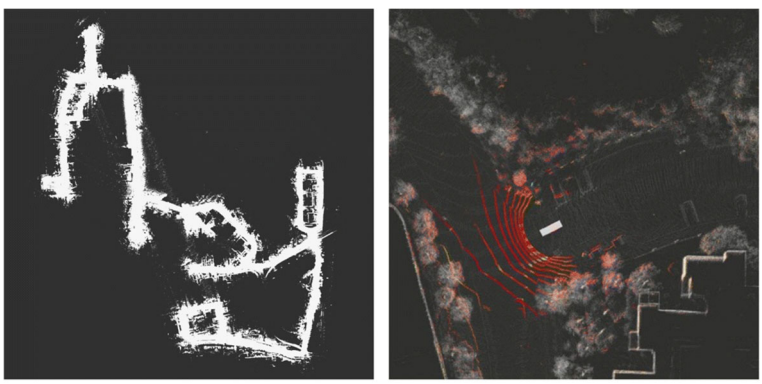
(d) Labeled LiDAR clusters for road features and markings using (c).

Fig. 7 Camera-LiDAR fusion with cluster classification. The bounding boxes on the left column (a and c) provide labels for the LiDAR clusters shown on the right (b and d) by using projection overlap

2.2.3 Global and motion planning

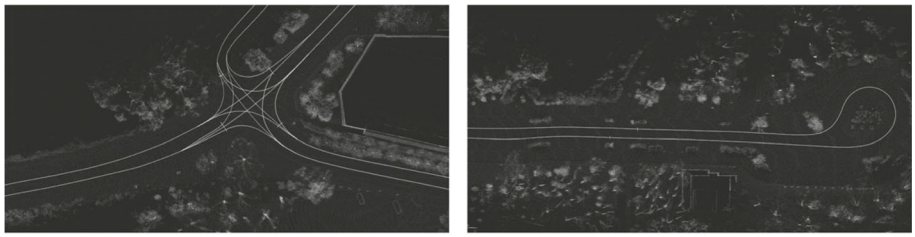
By understanding the pose of the robot over time, a global plan can be generated given a target location and the predefined road networks introduced in Section 2.2.2. This plan is represented by sequences of polylines and describes the trajectories that need to be traversed to reach the destination of interest. Given that this plan is defined within the same reference frame as the ego-vehicle pose, it can be utilized for motion planning and decision making by considering static map elements, and pedestrians and vehicles nearby. To facilitate this process, we employ a modified version of the OpenPlanner [23] introduced as part of the Autoware stack.

OpenPlanner consists of two core submodules which include a global planner and a motion planner. In the global planning phase, an optimal shortest path between the pose of the vehicle and a destination is calculated recursively if the endpoints within the road network



(a) (b)

Fig. 8 A downsampled point cloud map for campus (a) and localization using scan matching (b)



(a) (b)

Fig. 9 Vector/High-Definition (HD) maps displayed on top of the point cloud maps for two campus areas. We annotate centerlines, stop lines, crosswalks and speed limits

graph are connected; this is represented as a trajectory in Fig. 10(a). Finally, in the motion planner module, traffic rules and obstacle avoidance strategies are enforced while the ego-vehicle uses the reference trajectory provided by the global planner to reach its destination; the Finite State Machine (FSM) associated with this process is shown in Fig. 10(b).

To enable robust navigation while enforcing speed limits, following other vehicles, and making planned stops, various modifications were performed to the Forward, Follow, and StopSign states. This was a necessary step to ensure that the ego-vehicle would react independently of the road conditions and geometry. Initially, the OpenPlanner logic was designed with the assumption that most roads are planar thus simplifying the physical constraints.

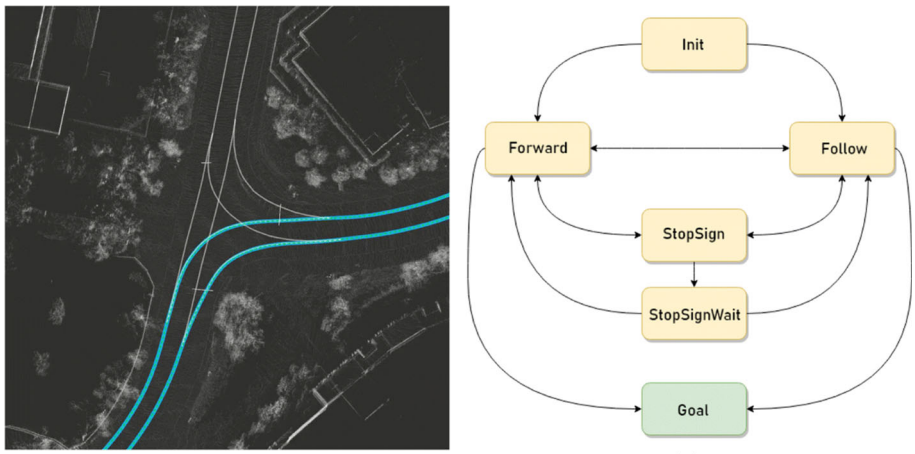
Speed Keeping

While the logic associated with speed keeping is straightforward, there are additional considerations performed within the Forward state of the FSM. This state specifically considers the vehicle's current speed, speed limits, and the DBW enable signal to generate a target speed. The target speed converges to the speed limit imposed by the road segment unless there is a maximum speed set for the entire mission. The logic associated with this rate of convergence is dictated by the acceleration rates a_{accel} and a_{decel}, shown in Algorithm 1; these are constants tuned for comfort in terms of m/s^2.

An important design consideration involves target speed relaxation when the DBW system is disengaged to prevent windup effects from the controller. This can prevent erratic behavior when the system is being reengaged and an acceleration or brake command is being set after large errors were accumulated within the controller. The trigger is shown within the first conditional of Algorithm 1. A visualization of the signal generated during the Forward state can be visualized in Fig. 14; where the red control signal up to timestep 93 corresponds to the output generated by our algorithm and the blue signal corresponds to the speed of the vehicle. Additional details on the design of the controller employed are covered in Section 2.2.4.

Obstacles and Planned Stops

The states that account for vehicle following and planned stops correspond to Follow and StopSign, respectively. In



(a) (b)

Fig. 10 Planned global trajectory based on vector map (a) and the Finite State Machine for the motion planner (b)

Algorithm 1: The target speed generated during motion planning is a function of the vehicle's current speed, previous target speed, as well as the DBW enable/disable signal to generate smooth and comfortable acceleration/deceleration signals.

Data: drive-by-wire enable signal **auto_enabled**, vehicle speed **v**, previous target speed **v_prev**
Result: target speed **v_target**

```
v_target = v;
if auto_enabled then
    v_target = v_prev;
end
if v_target > speed_limit + Δ then
    a = a_decel;
else
    a = a_accel;
end
v_target = v_target + a · dt
```

contrast to our approach for speed keeping within the Forward state, the estimated acceleration and target speed generated during vehicle following and planned stops is dynamic by nature and requires an accurate formulation to stop within a specific distance for planned stops or to match the speed of the vehicle in front of the ego-vehicle. Evidently, this depends on the conditions of the road and weight of the vehicle. While experiments were performed with obstacle avoidance maneuvers, due to limited evaluation and testing, this state was not incorporated in the initial version of our experiments. For this reason, this state is omitted from the FSM.

For both strategies, the decision-making process is initiated by identifying the distance to the waypoint of interest w_p. From Fig. 11, it can be observed that for vehicle following, w_p corresponds to the waypoint closest to the rear of the object of interest $w_{obstacle}$ (Fig. 11(a)) and for planned stops, w_p corresponds to the center of the stop line $w_{stopline}$ (Fig. 11(b)). Although the Euclidean distance between the waypoint closest to the front of the ego-vehicle (w_{ego}) and w_p can be estimated, this distance estimate may not be representative of the distance that the vehicle will ultimately traverse. Instead, the approach employed uses the complete trajectory to aggregate the pairwise distances between adjacent waypoints starting from w_{ego} (i) and ending at w_p (j); this estimate can be computed iteratively by Eq. (1)

$$d_{\text{target}} = \sum_{k=i}^{j} \sqrt{\left(w_{k+1} - w_k\right)^\top \left(w_{k+1} - w_k\right)}\,. \tag{1}$$

With the distance estimate (d_{target}), a linear kinematics approach is utilized to identify the acceleration rate needed to ensure that the ego-vehicle stops at the location of interest or reaches the speed of the vehicle ahead. The following expression approximates the acceleration a needed to reach the desired speed v_f within the distance d_{target} given the current speed of the vehicle v_{ego}. Furthermore, by noting that a planned stop is a special case when v_f is zero and for vehicle following v_f corresponds to the speed of the vehicle ahead, the same kinematics formulation can be applied.

$$\begin{aligned} 2a \cdot d_{\text{target}} &= \left(v_{\text{f}}^2 - v_{\text{ego}}^2\right) \\ \Rightarrow a &= \left(v_{\text{f}}^2 - v_{\text{ego}}^2\right) / 2 \cdot d_{\text{target}}\,. \end{aligned} \tag{2}$$

A final consideration in the design of our approach involves the state transition triggers. While the acceleration described above can be utilized to update the target speed v_{target} that will be delegated to our downstream controller, one must first determine the state transition logic to be able to enter the Follow and StopSign states. To formulate this trigger, we first characterize the average braking performance of our vehicles by performing a sequence of complete stops to measure the quality and braking performance under various scenarios including hills and flat roads. By measuring initial speed, final speed, and distance traversed, an estimate for average acceleration can be identified: a_{brake}. This constant can jointly be applied with equation Eq. (3) to approximate the distance required to reach v_f given the ego-vehicle speed v_{ego}. This distance denotes $d_{trigger}$ can then be used to trigger the state transition logic. The state transition logic is then triggered when $d_{trigger} > d_{target}$. As a final note that is specific to the StopSign state, once the vehicle performs a complete stop, the StopSignWait state is entered for three seconds are required by law, before continuing.

$$d_{\text{trigger}} = \frac{\left|v_{\text{f}}^2 - v_{\text{ego}}^2\right|}{2 \cdot a_{\text{brake}}}. \tag{3}$$

In practice, this approach performs well in a variety of scenarios including steep inclines with additional passengers onboard. A visualization of the speed signal generated by the planner for planned stop can be visualized in Fig. 14 after timestep 93, where the red signal corresponds to the target speed generated by the planner and the blue signal to the actual vehicle speed after our controller matches the target speed. The details for the design of our controller are covered in the next section.

2.2.4 Trajectory following and control

The final output from the motion planner consists of speed encoded waypoints that account for speed limits, stop lines, and obstacles ahead of the ego-vehicle. With this trajectory, we employ the Pure Pursuit path tracking algorithm [24] to estimate the radius of curvature needed

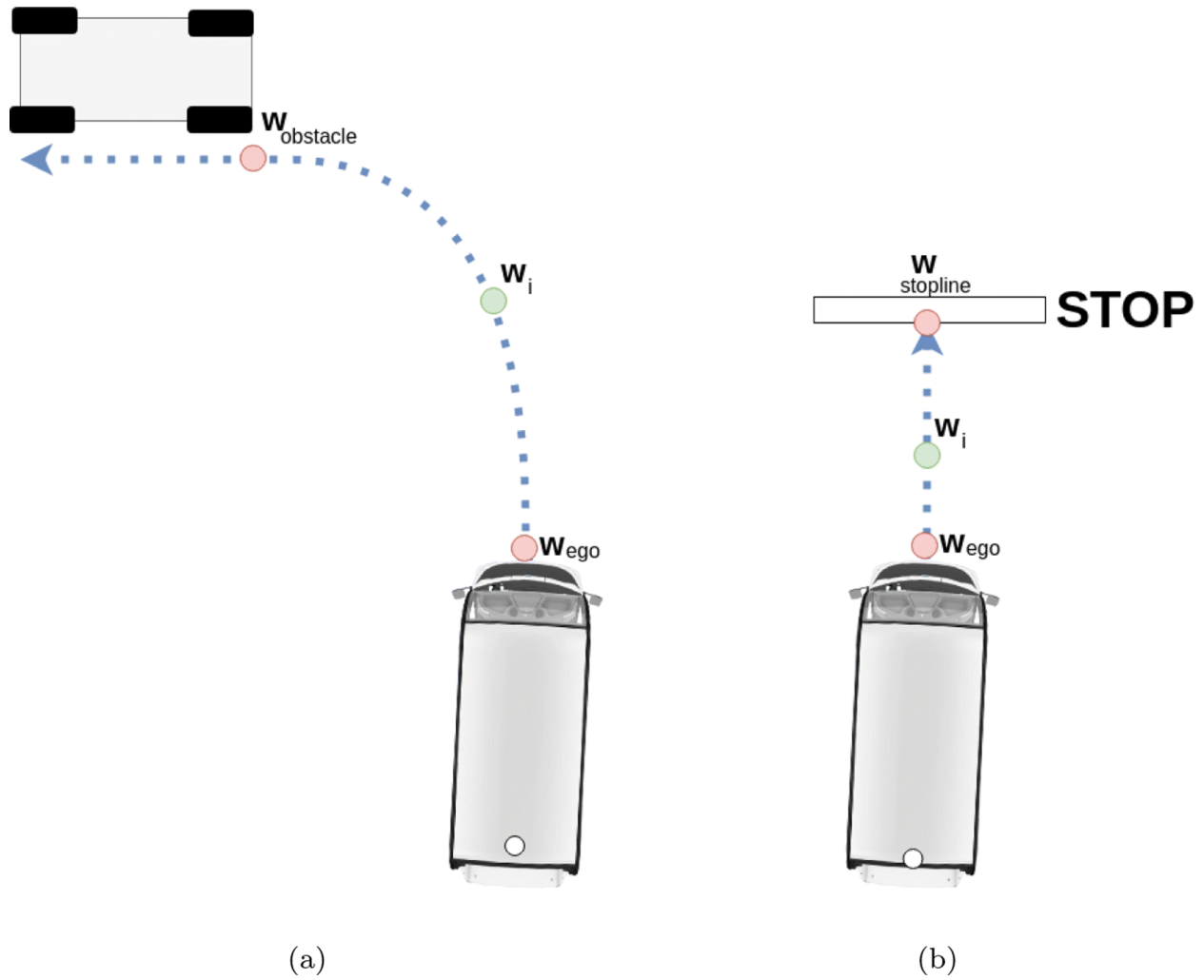


Fig. 11 The selection of the waypoint of interest w_p for vehicle following (a) and planned stops (b)

to stay on the path. Furthermore, the radius of curvature is used in conjunction with the bicycle kinematics model and the vehicle specific steering rack ratio to calculate the tire and steering angles of the vehicle. Finally, a Proportional Integral Derivative (PID) controller is employed to determine the accelerator and brake control inputs given the target speed set during motion planning.

Steering Angle and Angular Velocity

Pure Pursuit uses a simple tunable parameter referred to as the lookahead distance (l) to identify a moving target (x, y) along the trajectory. By following the geometric relationship shown in Fig. 12(a) and Eq. (4), the curvature of the arc $(1/R)$ that joins the moving target with respect to the rear axle of the vehicle can be estimated. This expression implies that the radius of the arc simply differs by a distance d from the x component of the target point. In practice, the lookahead distance must be tuned to achieve good performance. Small values of l can cause extensive oscillations in steering whereas larger values may reduce the error gradually.

$$\begin{aligned} &x + d = R, \\ &x^2 + y^2 = l^2, \\ &R^2 = y^2 + d^2 \\ &\Rightarrow R = l^2/2x\,. \end{aligned} \tag{4}$$

Additionally, to model the relationship between the curvature of the arc and the tire angle (θ_W) of the vehicle, the bicycle model is applied using vehicle specific dimensions (Fig. 12(b)). The rack ratio is then used to estimate the steering angle input for control Eq. (5).

$$\begin{aligned} &\tan\theta_W = \frac{L_W}{\sqrt{R^2 - L_{\mathrm{CW}}^2}}\,, \\ &\theta_s = C_{\mathrm{rack}}\,\tan^{-1}\left(\frac{L_W}{\sqrt{\left(\frac{l^2}{2x}\right)^2 - L_{CW}}}\right)\,. \end{aligned} \tag{5}$$

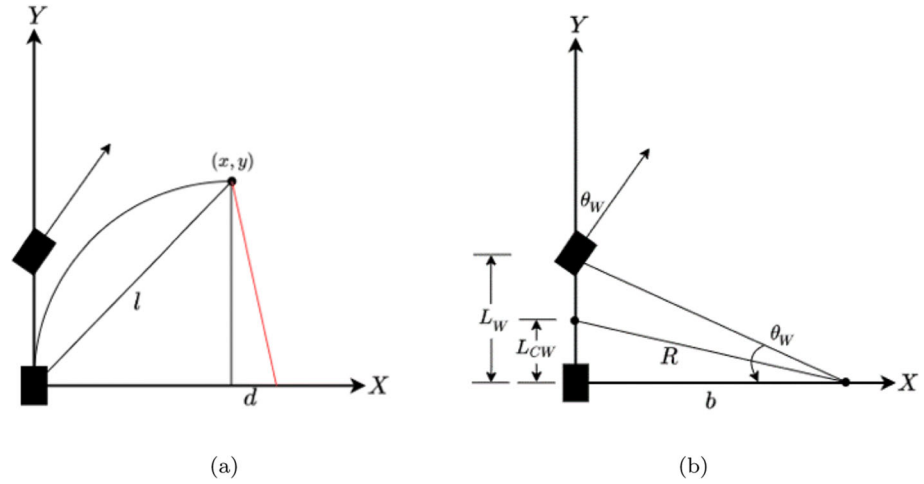


Fig. 12 Pure Pursuit (a) and bicycle model (b)

Before the steering control input can be executed, the velocity that corresponds to the estimated steering wheel angle must be determined. This is a secondary control input that is provided to the DBW system while executing the steering control input and must be characterized depending on the driving conditions. Our approach is modeled after the notion that minimal steering should be performed when the vehicle is driving at higher speeds. On the other hand, if the vehicle is driving slow, maximum flexibility can be provided to increase maneuverability. To model this relationship, the speed of the vehicle is used to formulate an angular velocity estimate using the vehicle-speed to max-speed ratio ($v_{\text{current}}/v_{\text{max}} \cdot \beta$) as shown in Eq. (6) When the ego-vehicle is operating at low speeds, this ratio approaches zero and maximum angular velocity is permitted. In contrast, when the vehicle is operating at speeds close to its limit, the ratio approaches one, and the angular velocity permitted approaches zero. In practice, β is simply a constant slightly larger than one to prevent a zero angular velocity and v_{max} is set to be 25mph (40.2 km/h); this speed corresponds to the maximum vehicle speed and is delimited by its manufacturer. Additionally, the campus speed limit is 25mph.

$$w = w_{\text{max}} \left[1 - \frac{v_{\text{current}}}{v_{\text{max}} \cdot \beta} \right] . \tag{6}$$

Acceleration and Braking

As previously discussed, the final output generated from the motion planner includes a desired speed limit in addition to the trajectory. While the trajectory waypoints are useful for estimating steering wheel control inputs, the target speed can be used to determine the accelerator and brake control depending on the error between the current speed and the target speed $e(t)$. In contrast to steering wheel control, the DBW receives unitless inputs for braking and acceleration. Thus, making it ideal for controllers such as the PID controller.

In this work, we employ two traditional PID controllers to estimate braking and acceleration control inputs. The iterative process of a PID controller is formulated using the error $e(t)$ between the target speed and the current vehicle's speed as reported by odometry. For acceleration, the error is defined by $e(t)_{accel} = e(t)$, whereas for braking, the error is its negative $e(t)_{brake} = -e(t)$. The overall process is shown in Fig. 13 and the control equation is shown in Eq. (7).

$$u(t) = K_{\text{p}} e(t) + K_{\text{i}} \int_0^t e(\tau) d\tau + K_{\text{d}} \frac{de(t)}{dt} . \tag{7}$$

Two additional considerations performed while implementing the PID controllers involve the windup effect and integral sum thresholding. During DBW disengagements, it is possible that the desired speed and the current speed diverge. As a result, this can lead to excessive integral sums if a maximum threshold is not set and the DBW system may set a maximum acceleration or brake command while re-engaging the system depending on if the error is positive or negative. To prevent this from occurring, the integral sum is reset to zero when the vehicle's speed is relatively close to zero and the maximum feasible control input is used to threshold and prevent the sum from reaching high numerical values. Finally, additional steps are taken in the motion planner: if a disengagement is reported by the DBW system, the planner will automatically reset the target speed to prevent large errors from occurring and increment or decrement gradually based on the current speed if the system re-engages. By thresholding the integral sum and informing the planner on the state of the DBW, this effectively prevents adverse

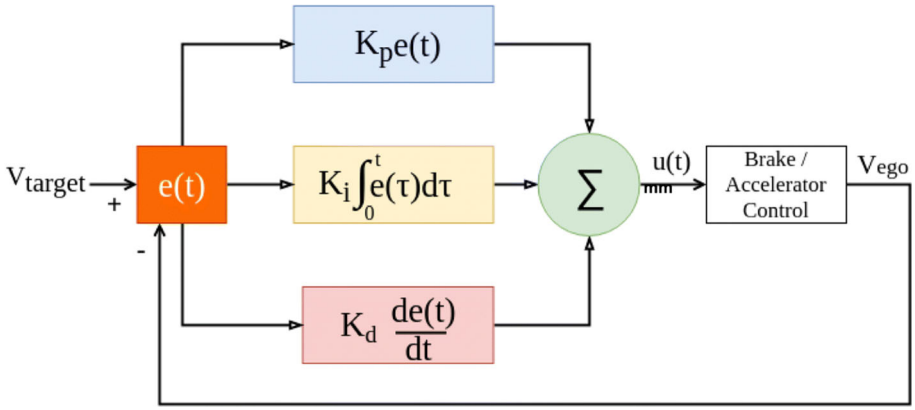


Fig. 13 The PID controller for braking and acceleration control

windup effects. A visualization of the ego-vehicle performing acceleration, speed keeping, and braking under the approach introduced is shown in Fig. 14.

2.3 Verification and early testing

An important aspect throughout the development, verification, and testing phases involves safety. To ensure that proper testing procedures were followed, our team developed in simulation using OpenPlanner and verified in non-public roads for six months. The ROS and OpenPlanner framework facilitated data-driven simulation to verify that the planner entered the correct navigation states given perception and localization information. Additionally, a simple kinematics model was employed to verify basic functionality such as map definitions and global plan generation. This approach facilitated the initial implementation and helped debug complex cases.

For live testing, safety precautions were considered by using an emergency stop button (Fig. 15) and by enabling direct manual override using the steering wheel or brake pedal. In the next section, our approach is evaluated in an autonomous mail delivery scenario. While our system was evaluated offline and in non-public roads prior to initial deployment, the safety drivers that participated in the pilot deployment participated in a training program provided by the UCSD Risk Management office. The deployment and testing associated with this pilot program was performed in collaboration and with approval of UCSD facilities, police station, and the mailing center.

3 Lessons from first long-term test

To gain a better understanding of the capabilities and overall robustness of our system, our team conducted a project on autonomous mail delivery over the course of four-months in collaboration with the UC San Diego mailing center. To facilitate mail delivery applications, one of the vehicles introduced in Section 2 was retrofitted with a ramp and hamper fasteners (Fig. 16). The field experiments were performed with the same safety driver throughout the deployment period between summer and fall 2019.

3.1 Disengagements from autonomous mail delivery

By fall 2019, our vehicle had engaged in a total of 89.9km or equivalently 6.9 hours in autonomous mode. As introduced in Section 2.1.2, various signals were recorded using a logging device including localization information, target speed, vehicle control signals for steering, braking, acceleration, and the DBW enable/disable signal. Secondly, the ROSBAG file format was used to record raw sensor data including image and LiDAR data. Finally, disengagement details were labeled by the engineer onboard of the vehicle for every intervention performed by the safety driver. The overall robustness of the system is characterized by using mean-distance between interventions and mean-time between interventions [25]. This is performed by using the DBW enable signal to separate the total distance traveled and total uptime in autonomous and manual modes. To further understand the dependability of the system on the safety driver, the manual and autonomous segments are separated as shown in Eqs. (8) to (11), where MDBI_A and MTBI_A measure the overall robustness of the system over time. On the other hand, MDBI_M and MTBI_M describe the average distance and time for which the safety driver performed manual interventions. This effectively normalizes the number of disengagements in terms of time and distance whilst simultaneously describing the dependability of the system on human drivers.

$$\text{MDBI}_A = \frac{\text{Total Auto Distance}}{\text{Number of Interventions}}. \tag{8}$$

$$\text{MTBI}_A = \frac{\text{Total Auto Uptime}}{\text{Number of Interventions}}. \tag{9}$$

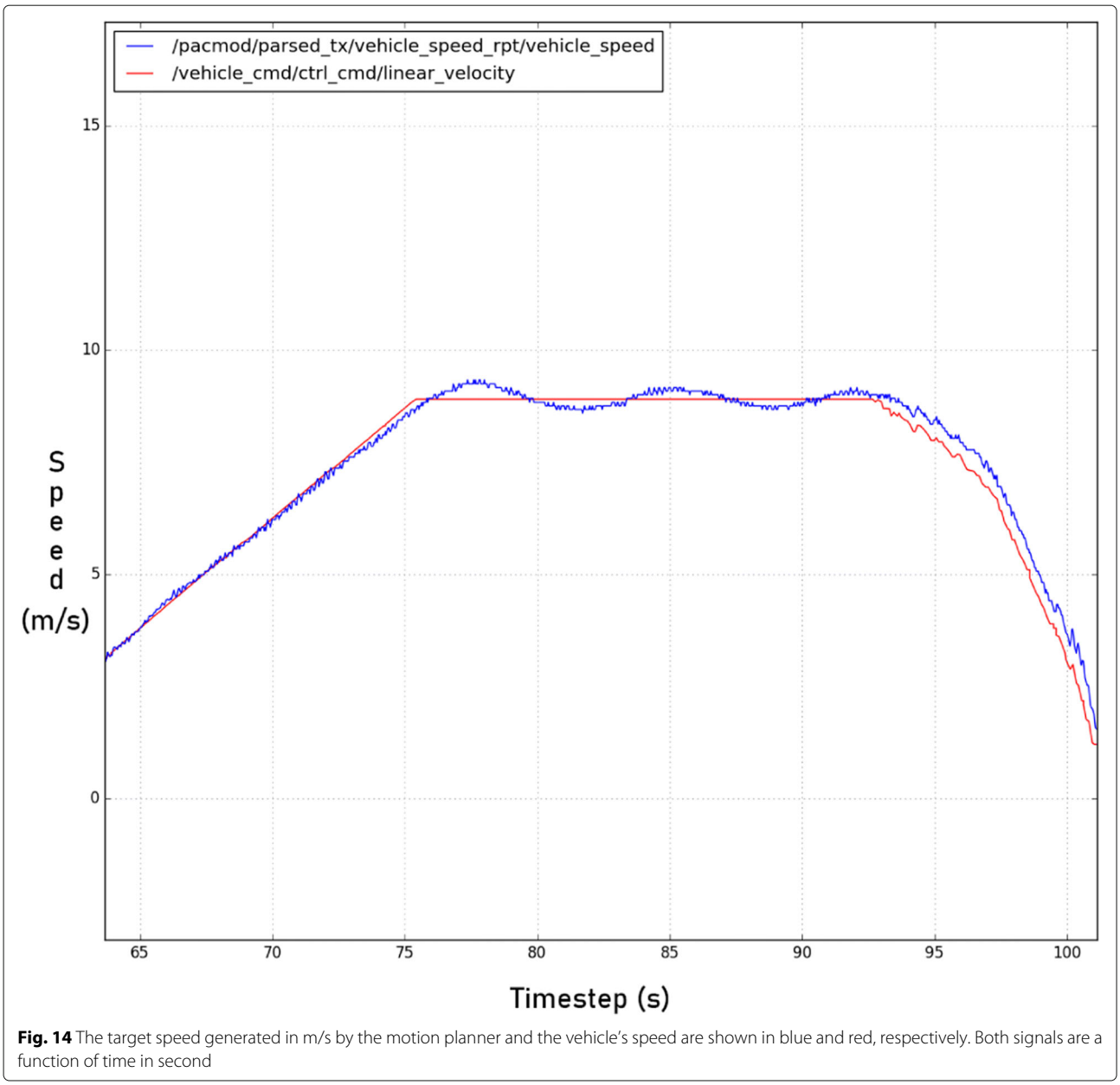


Fig. 14 The target speed generated in m/s by the motion planner and the vehicle's speed are shown in blue and red, respectively. Both signals are a function of time in second

$$\text{MDBI}_M = \frac{\text{Total Manual Distance}}{\text{Number of Interventions}}. \tag{10}$$

$$\text{MTBI}_M = \frac{\text{Total Manual Uptime}}{\text{Number of Interventions}}. \tag{11}$$

With the metrics introduced, the statistics for the summer and fall quarters are summarized in Table 1 and the overall across both quarters is shown in Table 2. From the overall disengagement statistics, one can infer that the vehicle drove autonomously an average of 380m between a disengagement, where each disengagement performed by the safety driver corresponded to approximately 23m of manual driving. This is equivalent to driving for 106s between a disengagement, where each disengagement corresponds to approximately 12 s of manual driving. Additionally, summer and fall quarter campus trends are reflected in the performance. During summer, the pedestrian and vehicle activity is considerably less compared to fall quarter. This explains the difference in terms of performance given the complexity. The disengagement reciprocals are also provided in each of the tables. Ideally, these values should approach a limit of zero; meaning that zero interventions were encountered.

Lastly, to gain a comprehensive understanding of failure modes, the concept of intervention maps was introduced to capture spatial dependencies and trends that MDBI and

Fig. 15 The emergency button for safety consideration

MTBI may be unable to capture. An intervention map incorporates the disengagement data into a 2D map representation of the routes traversed during autonomous navigation. This is accomplished by discretizing the state of the vehicle overtime with a $1 \times 1m^2$ grid resolution and when a disengagement occurs, the disengagement count for the closest matching cell is incremented. The intervention map that corresponds to our two-quarter mail delivery project is shown in Fig. 17. All disengagement grids shown in this figure are normalized based on the grid with maximum disengagement value to convey density.

3.2 Lessons

In this section, various lessons are drawn from the disengagement data previously introduced. While there are various engineering related efforts that can increase the overall robustness of our approach, we focus on the core limitations experienced that are active research efforts. This includes lessons on perception, scalability and scene understanding for robust navigation.

Fig. 16 The retrofitted vehicle for mail delivery

Table 1 MDBI and MTBI Disengagement Summary for summer and fall 2019

Summer 2019	**MDBI**	**MDBI^{-1}**	**MTBI**	**MTBI^{-1}**
Autonomous	414.201	0.0024	113.82	0.00878
Manual	24.00	0.0416	12.77	0.07829
Fall 2019				
Autonomous	283.08	0.0035	84.44	0.0118
Manual	19.25	0.0519	11.54	0.0866

3.2.1 Sensor fusion for detection

By introducing fusion-based perception methodologies in our software stack, additional contextual information can be provided for downstream tasks that depend on an object's classification and pose that is often a critical component for decision making and ultimately robust navigation. Nevertheless, in the processes of fusing features and objects across various sensor frames, considerable preparation overhead was encountered. This corresponds to the process of characterizing sensor properties: intrinsic and extrinsic calibration. As previously introduced, the former describes the camera projection properties and the latter the transformation between a sensor pair. However, this calibration process was performed manually and periodically to ensure that vibration did not generate severe compound errors during projection. This presents a hurdle in the development and testing process in terms of continuous maintenance. For this reason, online calibration systems for parameter estimation are of great interest; research directions in this area are explored in Section 4.1.

Another challenge encountered in the process of evaluating our perception system involves detection and tracking of large moving objects such as cars, trucks, and buses. As introduced in Section 2.2.1, our approach employs a clustering-based approach for object detection. While this technique works well for smaller agents such as pedestrians, the approach does not readily capture the centroid, shape, and orientation characteristics of rectangular objects. This can make state estimation a non-trivial task depending on the perspective of the object even if it is stationary.

As a step towards improved centroid and shape estimation, an L-shape fitting approach [10] was explored that leverages RANSAC to identify the best fit orthogonal lines given a vehicle cluster detection. This method assumes that at least two sides of a vehicle are detected

Table 2 MDBI and MTBI overall disengagement summary

Overall	**MDBI**	**MDBI^{-1}**	**MTBI**	**MTBI^{-1}**
Autonomous	380.42	0.0026	106.25	0.0094
Manual	22.77	0.0439	12.46	0.08028

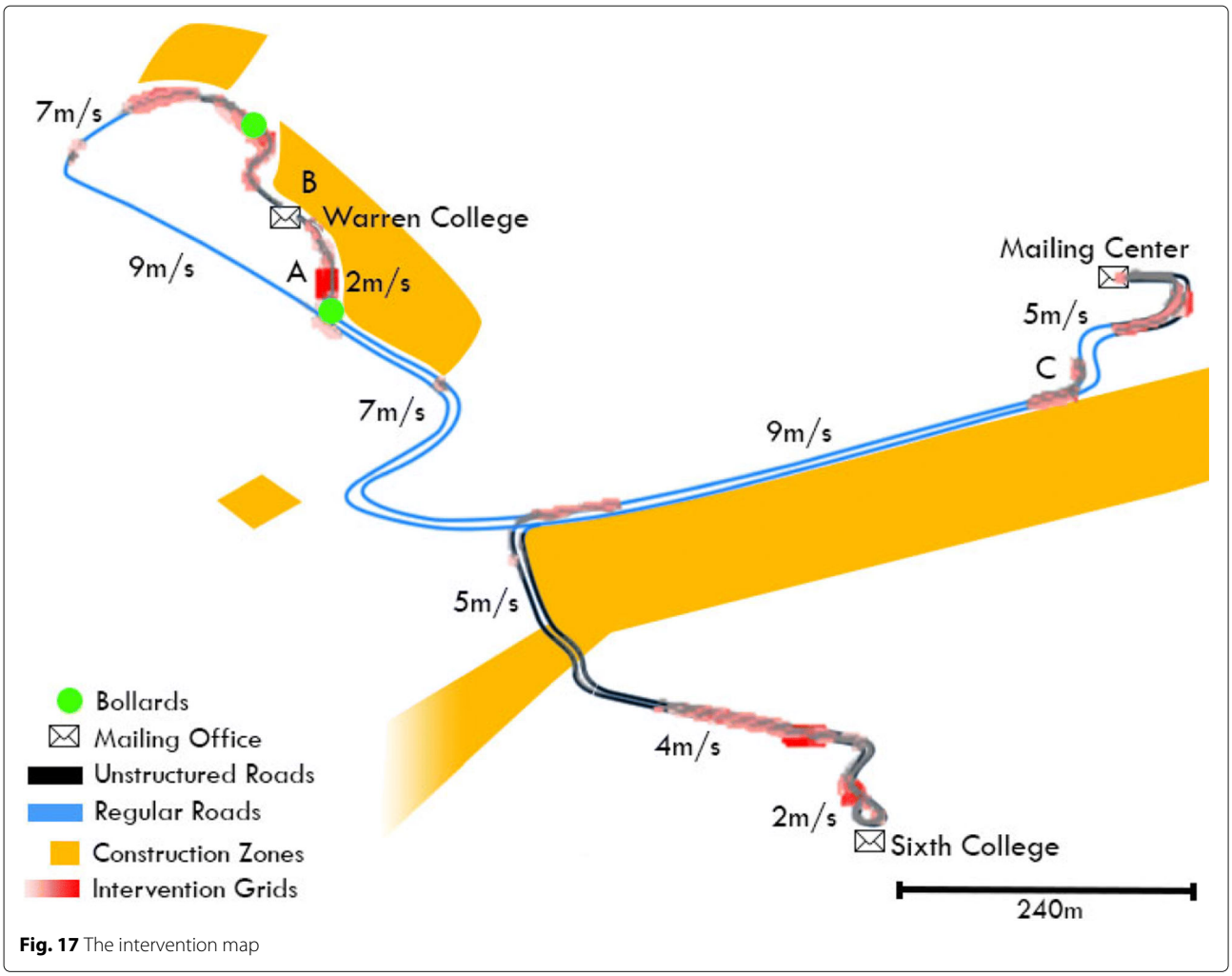


Fig. 17 The intervention map

from the LiDAR perspective and the longer side corresponds to the longitudinal axis of the vehicle. However, in practice this assumption again depends on perspective and may initialize a track with swapped lateral and longitudinal axes. This failure mode can be visualized in Fig. 18, where the L-shape detection approach incorrectly assigns the orientation of a vehicle with track B. In contrast, the pedestrian with track A is constantly detected and tracked as it crosses in front of the ego-vehicle. Given the importance of detection and state estimation for various agents in a scene, learning based methods for sensor fusion are explored in Section 4.2.

3.2.2 Long term forecasting, intent recognition, and interactions

From Fig. 17, it is evident that intersections are among the areas with higher number of disengagements. Additionally, road segments marked as Unstructured correspond to areas that are highly dynamic and often involve complex interactions with road users. In fact, prediction related interactions are among the primary reasons that disengagements were performed during our mail delivery missions. These specific interactions are further analyzed by reviewing the logs provided by the engineer onboard of the self-driving vehicle throughout the four-month deployment period. The scenarios considered part of this subset of disengagements involve errors due to inaccurate behavior forecasting of other agents in the scene that as a result propagated incorrect control actions by the planner and controller. From these logs, a subset of our original intervention map is generated as shown in Fig. 19, where the green regions correspond to areas with at least one disengagement related to prediction failures. Evidently, intersections are among the areas with higher failure rates given that these often involve negotiating right-of-way.

An important note involves recent developments in prediction. Even though prediction has been studied extensively in recent years in the form of trajectory prediction including methods that leverage Generative Adversarial Networks (GAN) [26–30], Conditional

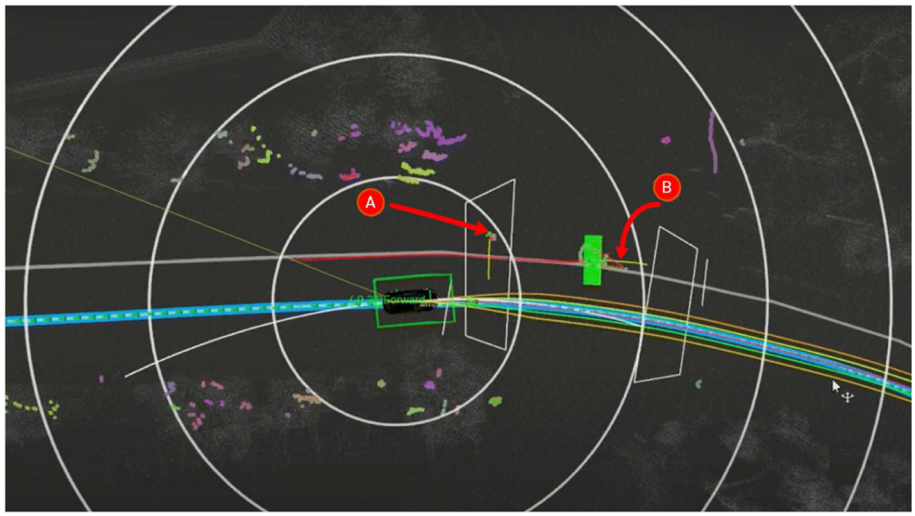


Fig. 18 The clustering-based method works well for pedestrians (A) but not for vehicles (B). L-shape estimation might flip the edges when an observation is incomplete. In the figure, agent tracks are represented by yellow trajectories

Variational Autoencoders (CVAE) [31, 32], Graph Convolutional Networks (GCN) [33], and Self-Attention [34], less efforts have been made to understand intent and interactions that can potentially involve communication. An example is illustrated in Fig. 20, where two cyclists are negotiating the right-of-way with another vehicle agent and the safety driver of our self-driving vehicle. Since gestures are a natural way of conveying intent, there are design considerations when it comes to the development of intelligent vehicle systems that operate with or without a driver. In the example highlighted, the cyclist assumes that our safety driver has control of the vehicle leading to a gap between the design of prediction strategies and execution.

In addition to reducing the gap between the design of prediction strategies and execution, additional research directions involve interactions with law-enforcement, emergency vehicles, and construction personnel. An example of an interaction with construction workers in a dynamic environment is illustrated in Fig. 21 that corresponds to images captured from the front and side-view cameras during a data collection mission. In this scenario, a construction worker is controlling two-way traffic along a closed lane. From a navigation perspective, the system must revise its current plan by dynamically generating a new trajectory to account for the traffic cones, whilst simultaneously following instructions from the construction workers: the construction worker shown overrides

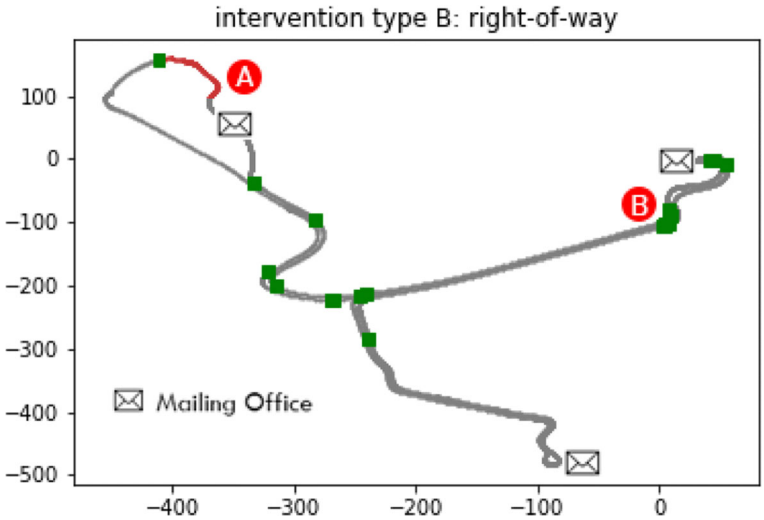


Fig. 19 Prediction related interventions (in green)

(a) (b)

Fig. 20 Example of interactions between a bicyclist and other vehicles (left), and between bicyclist and safety driver (right). This intersection corresponds to location B in Fig. 19. The images were captured by the forward facing cameras from the ego-vehicle

the existing navigation plan that entails making a stop at the intersection by using a 'SLOW' traffic sign followed by a hand gesture to wave the vehicle through. While it may be possible to define additional states for construction logic and employ learning-based techniques as part of a state-based planner, ensuring that the approach will generalize outside of the training distribution presents a new set of challenges.

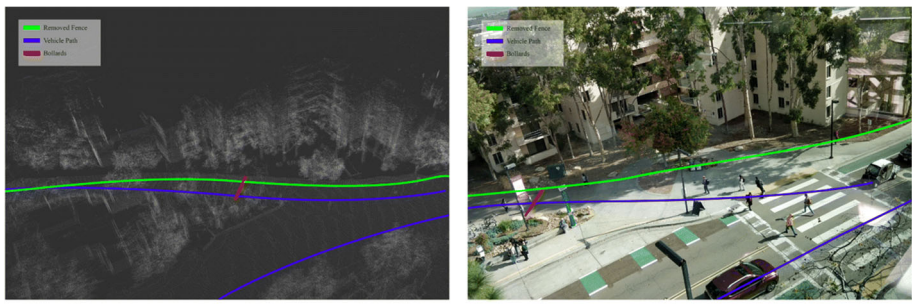
3.2.3 Static maps in dynamic environments

As previously introduced, our design strategy for navigation leverages road network definitions annotated at a centimeter-level that involved an extensive annotation process. While this HD map approach simplifies the complexity of trajectory identification for navigation, it presents several limitations in terms of scalability in dynamic environments. Since the annotations assume a static environment, drastic changes such as construction sites can impact the original definitions.

To provide further context, two examples in which the original map was affected by construction are shown in Figs. 22 and 23. Figure 22 corresponds to a mailing route that extends from a main road onto a large walkway; this route was shifted by a construction fence at the time the trajectory was generated. Not only do these changes imply that a new trajectory must be redefined as a result, but they can additionally impact the robustness of the localization algorithm given that the pointcloud map may contain missing features. A similar area that was affected by construction corresponds to Fig. 23, in which a three-way intersection along a mailing route of interest was converted into a round-about. For this scenario specially, the vehicle would be unable to navigate given the drastic changes to the road network and a new definition would be needed as well. While the UC San Diego campus covers a smaller area compared to large scale autonomous driving applications, the dynamic characteristics of urban environments present a scalability challenge that must be further explored. This is an area of interest that is investigated in Sections 4.3 and 4.4 by introducing an approach for dynamic trajectory generation without reliance on complex HD maps.

(a) (b)

Fig. 21 Example of an interaction with a construction worker (b) that directs opposing traffic using a single lane near a construction site (a). Navigating in complex environments requires dynamic methods that may not align with prior maps



(a) (b)

Fig. 22 A fence (in green) was removed and the route (in blue) needs to be adjusted

3.2.4 Closed loop simulation and testing

Although the simulation strategies employed can provide aid in early development and assist during case specific improvements, there is a gap between simulation and the real-world as well as scalability implications for data-driven methods. For fully simulated methodologies, accounting for non-linearities from real-world interactions becomes a non-trivial task and may not be a realistic approximation; examples include rendering photorealistic images and even simulating LiDAR data. While data-driven methods may be an excellent alternative, the approach quickly presents scalability constraints when small perturbations to the original data are needed or additional cases within the same category are of interest.

This presents a new direction in terms of simulation for reducing the gap between simulated representations whilst simultaneously increasing the scalability. Although there are more considerations such as generating realistic agent-to-agent interactions, we begin a search for different simulation strategies in Section 4.5.

4 Addressing the challenges

To address the challenges discussed in the previous section, we study further into these problems. In this section, we present our research in automatic calibration, sensor fusion, semantic mapping, dynamic trajectory generation, and building a digital twin.

4.1 Automatic calibration

One of our greatest challenges was posed by calibration. The calibration process provides intrinsic parameters of the camera and extrinsic parameters of the relative transformation between camera and LiDAR. These are fundamental for the detection, mapping tasks, and our sensor fusion approach. Both of the parameters need to be calibrated frequently as intrinsics change with temperature and extrinsics are affected by vibration. We start with a calibration board for intrinsic calibration and a hand-picked association based method for extrinsic calibration. These approaches are time-consuming and labor-intensive. Accordingly, efforts have been made to design automatic or semi-automatic calibration methods. In this section, we present our approach for intrinsic auto-calibration using traffic signs and extrinsic semi-automatic calibration with a black board.

4.1.1 Camera intrinsic auto-calibration

Camera intrinsic calibration is essential to relate image pixels with 3D feature points in the camera frame. For the applications, the accuracy of camera intrinsic parameters

(a) (b)

Fig. 23 An intersection (a) was converted into a round-about (b)

determines the performance of depth estimation, speed inference, and scene reconstruction.

There have been mature algorithms for estimating camera intrinsic parameters. For example, Zhang's method based on checkerboard [11] is widely used. Although automatic calibration for intelligent vehicles on the road is still an unsolved problem, we first question, is continuous automatic calibration necessary? From a 4.69*mi* drive around the UC San Diego campus, the results indicate a variation of 6.5% for the intrinsic parameters. Thus, making automatic calibration necessary.

One key challenge in auto-calibration lies in identifying adequate 2D-3D correspondences. Previous research efforts have focused on retrieving or constructing structures in the environment with known dimensions. For instance, road lines or carefully designed grid markings are common choices from which corners or vanishing points are extracted [35–37]. However, these structures are either variable in sizes and vulnerable to external influences, or are uncommon in urban environments. It is therefore reasonable to require that the reference, from which the correspondences are generated, has ubiquitous existence and standard size. Initially, we studied vehicle license plates, but vast experiments found large calibration errors with license plate points extracted from KITTI [38] dataset's video recordings. It was observed that license plates were small and blurry in images (due to vehicle motion); therefore, they introduced lots of uncertainty. On the contrary, we find traffic signs everywhere; these are relatively large and remain stationary. Before leveraging these references, we measured their distribution in a typical urban area to ensure sufficient features are present.. Figure 24 shows the distribution of stop signs along a UCSD campus loop. There are 43 stop signs over a distance of 7.47*mi*, and on average 195.5 detections/mi were estimated by using a benchmark object detection model [39]. As it is later verified, the amount of stop signs along can achieve frequent and accurate auto-calibration.

Experiments showed that the estimation accuracy of 2D feature points plays an important role in getting reasonable intrinsic parameters, and often a deviation unobservable by human eyes can lead to absurd calibration results. [40] suggests that line segments can be used to compensate for the pixel position noises. Therefore, we designed a pipeline to extract sub-pixel accurate corner points of traffic signs and use them for auto-calibration. As a concrete example, all our studies were based on stop signs. Starting with an image frame, we detect and crop out stop signs (if any) with a deep neural network using Mask RCNN [41]. We then apply a modified canny edge detector [42] to get the octagon contour. These contour points are processed by a RANSAC algorithm repeatedly, fitting eight line functions as the eight edges. If these are intersected in order, the estimated corner points achieve sub-pixel accuracy. Unlike 2D points, relative 3D coordinates can be looked up directly from the government's traffic sign manual. Finally, we complete the automatic system with Zhang's planar object calibration method.

During applications, we collect sequences of point pairs and calibrate one batch at a time. The two processes - data collection and batch calibration - can be asynchronous, and thus achieve real-time speed. It is, however, not straightforward to determine the batch size: a bigger batch will produce better intrinsic parameters (proven by experiments), but require more storage and longer time interval between updates. As an alternative, we introduced a Kalman filter to incorporate temporal data, and to carry out continuous updates.

This pipeline worked well in general, but sudden large errors appeared occasionally. While, these errors were reduced by the Kalman filter as more data was collected, this was because the estimated corner points deviated from the ground truth positions due to bad illumination. Therefore, we proposed two solutions for increasing the system robustness. First, after fitting the eight edges, the line positions are refined locally according to the image gradient in the perpendicular direction. Second, we check the shape formed by the eight fitted lines and make sure it is an octagon. These extra steps proved to be effective for speeding up the convergence of calibration updates.

The final system is shown in Fig. 25. Although stop signs are used to characterize the calibration process, the system can be easily adapted to various traffic signs or landmarks. More details can be found in [43].

A stop sign dataset was also built for experiments. The images were collected by one of our experimental autonomous vehicles (Fig. 1(b)) when driving through UCSD campus. We stored PNG images to preserve high-quality compression. Two cameras onboard of the vehicle were used and each detected 1,507 and 1,330 stop sign candidates using Mask R-CNN, respectively.

We compare our results with the ground-truth intrinsic parameters (GT) obtained from checkerboard calibration beforehand. The selected scale-free metric is the relative error of each focal length Eq. (12). It should be noted that for both of the stop sign candidate sets, the line refinement module was turned off. Experiments showed that the line refinement was helpful when the detected stop signs were fewer. Thus, line refinement is still necessary if the system is running in a region where the stop signs are rarely detected. The system selected 444 (Camera1) and 844 (Camera2) candidate stop signs ready for calibration. System results with Kalman Filters are shown in Figs. 26 and 27.

$$\text{Relative Error of} f = \frac{f - f^{GT}}{f^{GT}}, \tag{12}$$

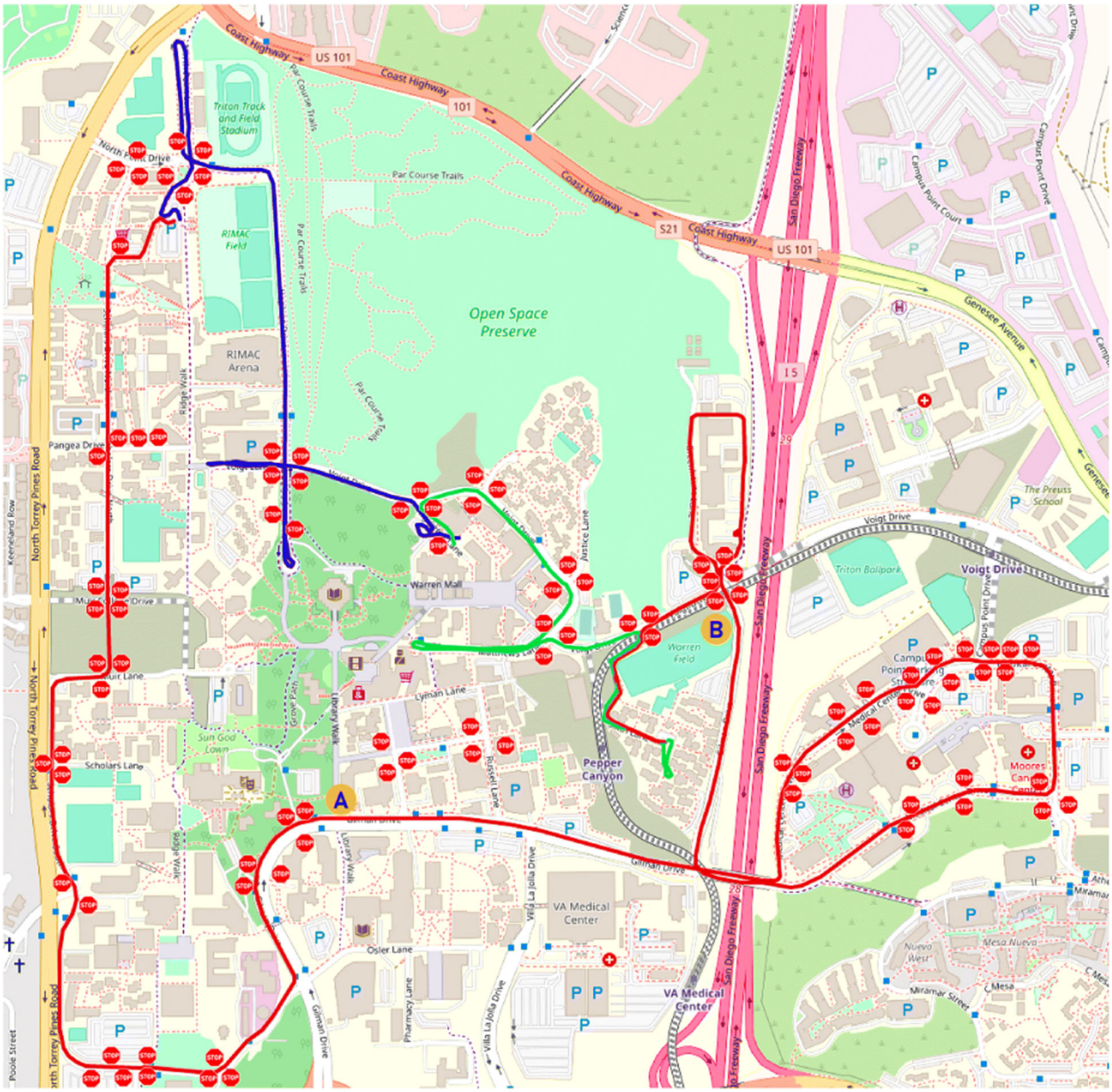


Fig. 24 A demonstration of a campus loop and the stop sign distribution. The blue, green, and red trajectories correspond to the three routes the vehicle drove in sequence

Our approach converges to the correct parameters as shown in the figures. A 5% relative error is achieved for the driver-side camera. We also intuitively demonstrate the effect of the relative error using a simple example of distance measurement. Suppose the principal point is fixed at the center of the image plane. If an algorithm measures the distance between an object and the camera, which is calibrated using our system, the measurement error (assuming the algorithm itself does not introduce additional error) is equal to the relative error of the estimated focal lengths. Take the driver-side camera as an example, an object estimated at 50m away will be located between 47.62 and 52.63m ($\frac{50\text{m}}{1\pm5\%}$). This error range is acceptable for safe reaction in urban areas with a 25mph (11.175m/s) speed limit: emergency braking experiments from 25mph to 0mph indicate that on downhill roads, 27.7m (4.4 s) are needed during emergency stops using our vehicles.

4.1.2 Camera-LiDAR extrinsic semi-auto-calibration

Camera-LiDAR extrinsic calibration calculates the transformation between LiDAR and camera. This transformation is important in sensor fusion tasks. For example, in

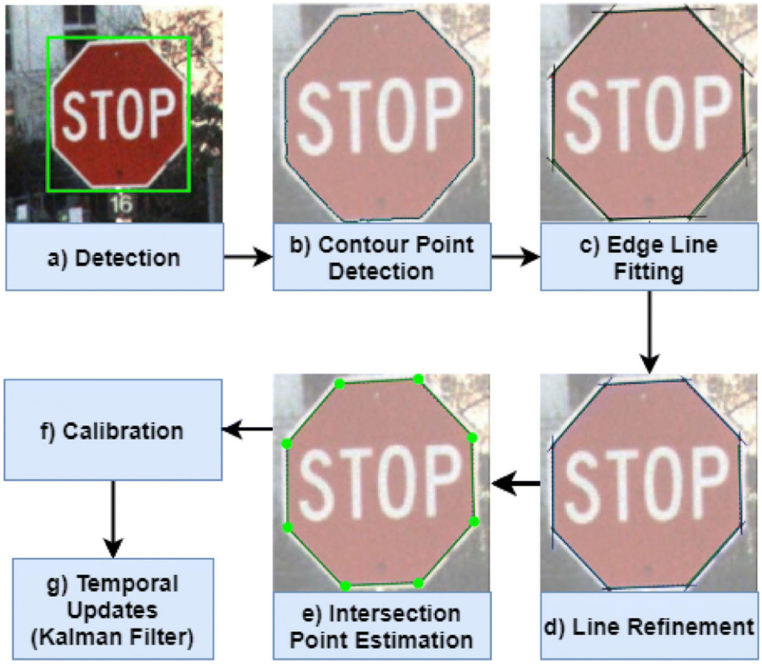


Fig. 25 A demonstration of the camera intrinsic auto-calibration framework

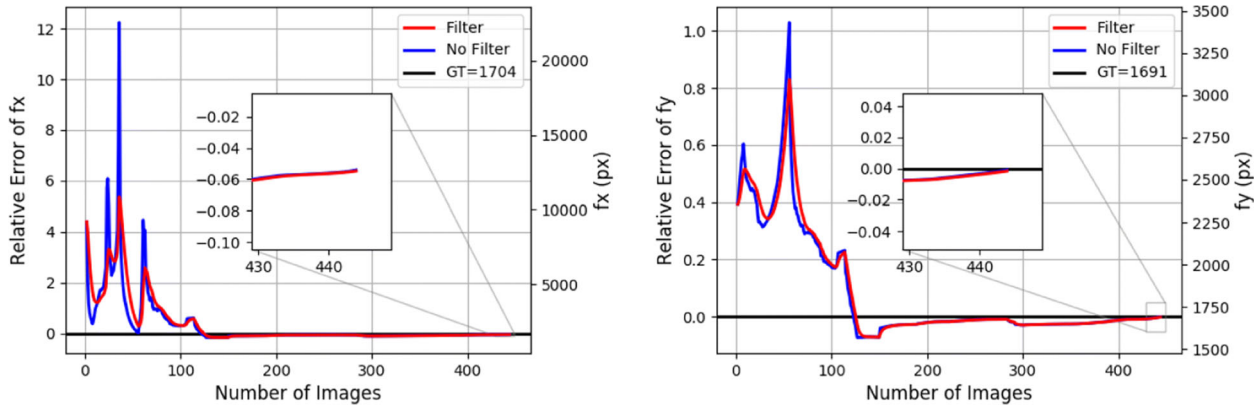


Fig. 26 Calibration Results of Driver-side Front Camera

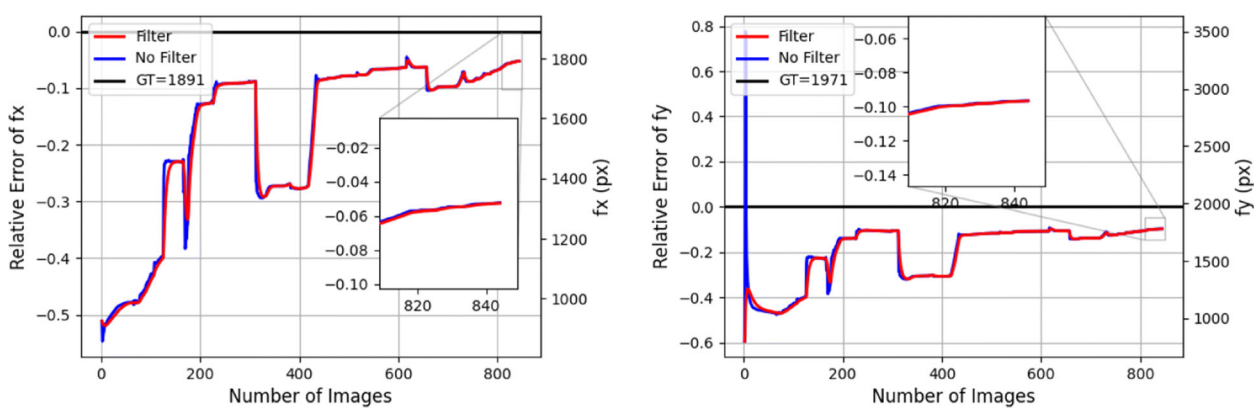


Fig. 27 Calibration Results of Passenger-side Front Camera

low-level fusion, the transformation is needed to project the LiDAR point cloud into the image to retrieve its color. In high-level fusion, the transformation is necessary to match objects detected in an image frame with the objects detected in the LiDAR data.

The calculation of the transformation is a straightforward problem. The challenge lies in specifying the corresponding features from the two sensors. Extrinsic calibration methods can be broadly divided into target-based or targetless approaches. While targetless approaches are more flexible, they often still have requirements of the environments, such as visual or geometrical textures [44]. Hence we pursuit the target-based approach. Such targets are usually hand crafted, such as chessboard, sphere [45] or board with AR tags [46]. We use a simple black rectangle board that is easy to obtain for this process.

Some approaches involve manual feature selection. For example, in [46], LiDAR points are manually grouped into edges so that corners can be calculated. The open-source tool provided by Autoware[2] we used initially also requires manual work. The tool provides a visualization for the point cloud and image frames, and requires us to manually select the corresponding points. We need to align a special landmark with the point cloud to make sure that we can tell the correct pair of points. Since the LiDAR point cloud is sparse, the alignment is not easy. Furthermore, to ensure the robustness, we had to repeat the process for more than 9 point pairs for various distances, which took a long time. What's worse, the transformation may change after one run of the vehicle due to vibration, the calibration takes up most of the time during our field-testing. Hence, we were in urgent need of designing a process to reduce the time for extrinsic calibration. The work closest to our approach is [47], where they also only requires a plain board. They use line detectors for images and the point clouds and formulate an optimization problem with a set of constraints.

It is relatively easy to detect 2D features in images, such as corners and edges. This is not the case for LiDAR. The Velodyne VLP-16 we equipped our sensor suite with is very sparse vertically; thus, it is blind to horizontal edges and points. Instead, we use rotated boards without horizontal edges for calibration, similar in [46]. As shown in Fig. 28, this is achieved by rotating the board to 30-60 degrees, so that all four edges have multiple intersections with horizontal lines. These intersections can be automatically picked from the data. With enough intersections along an edge, we can calculate the edge equation. Given all four edge equations, we can then calculate four corner points in 3D and the corresponding corner points in 2D can be calculated from a canny edge detector, followed by intersection points of edges. For a pair of image and point cloud data, we can extract at most four corresponding points, and thus we move the board around to increase the data pairs for robustness.

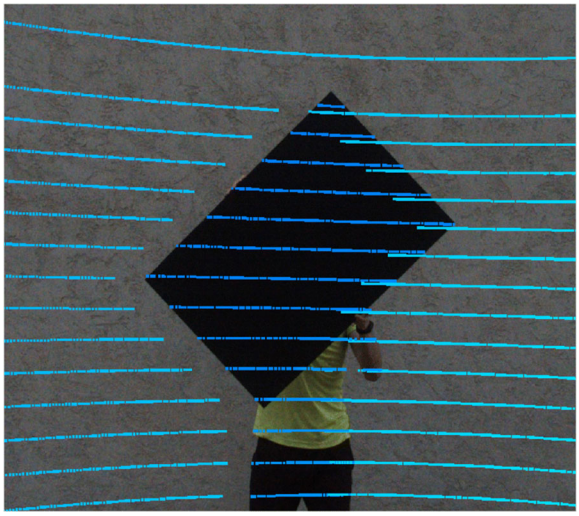


Fig. 28 One of our team members holding the board in the calibration process. LiDAR points are projected onto the image using the calculated extrinsic parameters. The color of LiDAR points is based on distance

In practice, we calibrate the system using a black board against an off-white wall to make sure the edge detectors work without interference from the background. We move the board to various poses in the intersection of the field of view of the camera and LiDAR. An ROS node runs the calibration program repeatedly until we are satisfied with the number of point pairs and reprojection error. A typical calibration process involves more than 60 point pairs that can be collected within a minute. Since we can easily get more point pairs than we need, a RANSAC variation m-estimator sample consensus (MSAC) [48] was used to improve the robustness.

Figure 28 shows our team member holding the board in the calibration process. After obtaining the extrinsic parameters, we color the LiDAR points by depth and project them into the image based on the estimated extrinsic parameters. The data collected for calibration can also be used to visualize the calibration results qualitatively. As we can see, the LiDAR points align well with the board. Typically, a reprojection error between 2 to 4 pixels is obtained. For reference, a 3px error for our camera data, with 1920px in width and a 60° horizontal field of view, means less than 0.1 degree error. For an object at 50 meters away, this could result in 8 cm error, which is acceptable. However, we later found that this reprojection error could be misleading about the actual performance of the system.

[2] https://github.com/Autoware-AI/autoware.ai/tree/1.8.0/ros/src/sensing/fusion/packages/autoware_camera_lidar_calibrator

The system sometimes demonstrates much larger errors than expected. We determined that the data collected occasionally posed challenges for the optimization. As shown in Fig. 29, where the ground truth camera location is given by the gray box. When we move the board around, the depth range can be limited by the sparsity of LiDAR scans. If we go too far, sufficient LiDAR scans may not intersect with the board's edges. Now, suppose our movement is near line 2 and matched key points were identified. Due to the error in the key point localization, we might finally estimate the red camera pose shown in dash lines. This estimated pose, despite being off both in translation and rotation, gives a low reprojection error for keypoints on line 2. But the error gets much larger for a longer distance, such as for points on the line 1.

Ideally, this could be resolved by collecting data for various distance ranges. But as we mentioned, the sparsity of LiDAR rings can limit the distance. When no larger board is available, we can perform RANSAC multiple times and pick the result that visually aligns well for various distance ranges. Alternatively, we consider the fact that the optimization challenge comes from a drift both in translation and rotation that reduces the reprojection error. Since the camera has rather low translational movement after vibration, we can fix the translation and only optimize the rotation.

4.2 Sensor fusion for robust tracking

Our sensor configuration implies that sensor fusion is necessary for robust tracking. The sparsity of the Velodyne VLP-16 LiDAR makes it challenging to detect objects beyond 30 meters robustly. This is especially hard for tracking vehicles. The clustering-based method [49] does not reflect the true center of vehicle and sometimes causes sudden shifts. On the other hand, the L-shape estimation approach discussed in Section 3.2.1 was not able to perform robustly for challenging scenes when there are only a few points projected on the vehicle. Currently, the state-of-the-art perception methods are heavily based on deep learning [50], [51], which generate much better predictions for the centers and dimensions of bounding boxes. However, they require a large amount of labeled data that is not provided for our collected data. For this reason, we investigate related topics on open-source datasets. The nuScenes dataset [21] was chosen as it includes radar, which provides potential for radar related fusion research.

Fusing multiple complementary sensors has been a popular research topic in autonomous driving. They are expected to improve the robustness and reduce the cost [52]. Fusion can happen in various stages [53]. Low-level fusion often involves highly coupled system design. For example, objects detected in cameras retrieve its depth from LiDAR data [54], or projecting LiDAR data into camera to get its color [55]. High-level fusion are modular at the expense of loosing some information. In [56], LiDAR, radar and camera objects are detected first then fused with an Extended Kalman Filter (EKF). More recently, deep learning based fusion achieve great performance [57], [58], [55]. But these black box models are hard to interpret.

Our first attempt was on fusing the LiDAR detected 3D bounding box with radar detected point targets using an EKF based approach [56]. We conducted experiments on a small subset of the nuScenes dataset for cars only. And evaluate based on the Average Multi-Object Tracking

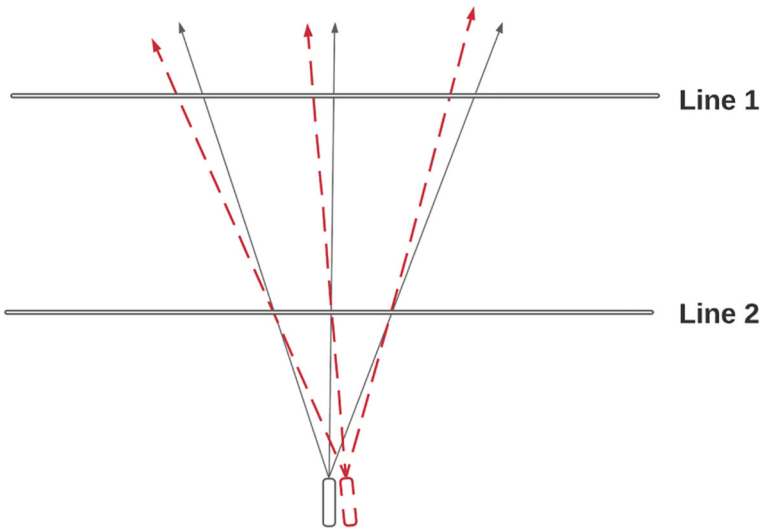


Fig. 29 The two camera positions has their keypoints match well on the line 2 but mismatch for the line 1

Table 3 Tracking results of the LiDAR and radar EKF based fusion

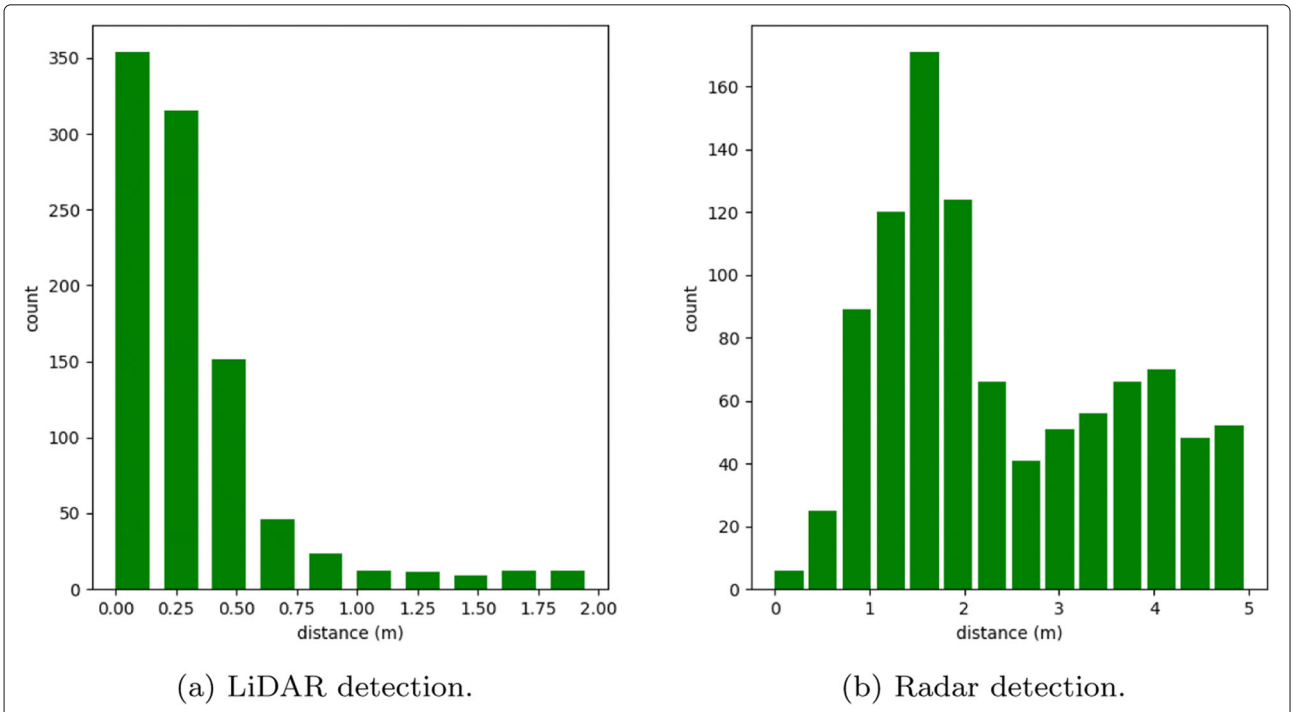
Modality and fusion method	AMOTA↑	AMOTP↓
v0: LiDAR	0.4957	1.0021
v1: LiDAR + radar update state	0.4460	1.2872
v2: LiDAR + radar update score	**0.5091**	1.0020
v3: LiDAR + radar update hits	0.4992	**0.9675**

Accuracy (AMOTA) and Average Multi-Object Tracking Precision (AMOTP) [59]. We follow the tracking-by-detection paradigm and use the detection from [51]. We then implement the LiDAR-based tracking following [60], and incorporated radar to the tracker. However, as shown in Table 3, adding the radar to update the state of the EKF trackers (v1) decreases the performance of the tracker. This contradicts our intuition that adding information can only be more helpful. Data analysis was performed to investigate the issue.

The result lies in the fact that radar data does not reflect the center of the object. As shown in Fig. 30, the LiDAR detection is like a bell shape curve around the ground truth. However, this is not the case for radar. The peak of the curve for radar was one meter off the center. The Extended Kalman Filter measurement model assumes the measurement is a Gaussian distribution around the ground truth. Given that the radar data does not satisfy this assumption, we observe that the result becomes worse when adding the radar.

Given that the radar objects is not informative about the center location, a follow-up question would be, does it reflect the existence of the object? State-of-the-art LiDAR-based object detectors often produces a lot of false positives. Hence, it would be valuable if radar objects can help eliminate some false positives. During our experiment, we determine that in some cases radar helps improve the accuracy of tracking by providing existence information, such as confidence score or hits (times of measurements being matched to a track). For example, when a radar detection is matched with an existing track, we update the confidence score of the track (v2) or update its hits (v3) to prevent it from being eliminated. Compared to direct state fusion that decreases the performance, these strategies increase the performance. However, the increment was negligible. This is because the radar only provides weak evidence of existence. As shown in the Fig. 31, many cases there are ground truth annotations but no radar objects. And given that track management system already eliminates most false positives using temporal consistency, this weak evidence from radar cannot contribute much.

Based on our experiments, incorporating radar measurements from nuScenes does not contribute much to the system. Similar results was also reported in [21], where learning based methods were explored. However, this does not mean that radar systems have no value for autonomous vehicles. Notice that our evaluation has a limited range up to 50 meters. This is a short range where



(a) LiDAR detection. (b) Radar detection.

Fig. 30 Detection center to ground truth center distance distribution for cars

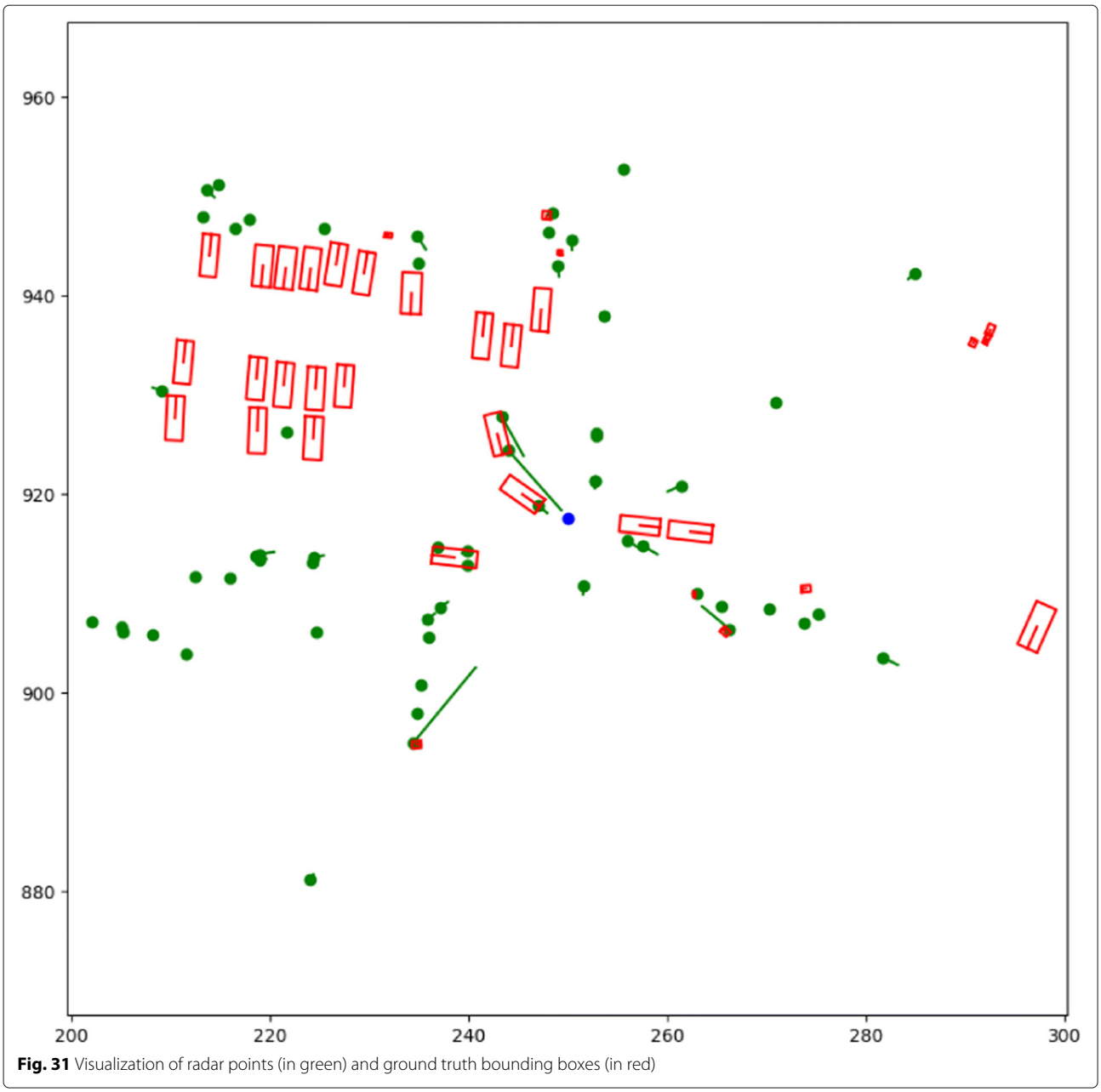


Fig. 31 Visualization of radar points (in green) and ground truth bounding boxes (in red)

LiDAR point cloud can be dense enough. But the value of radar is its long range perception capability, which cannot be evaluated based on the current dataset (which is annotated based on LiDAR). This is a common challenge for radar-based perception systems. Additionally, radar technology is also receiving improvements in terms of spatial resolution and signal-to-noise ratio.

4.3 Semantic mapping

High-definition (HD) maps can provide detailed structured information about the environment such as road topology, crosswalks, and stop signs. Many current autonomous driving systems depend on those detailed maps to help vehicles reason about the surrounding road elements and to make intelligent decisions accordingly. However, given the fast changing nature of the environment, maps become easily outdated, and it becomes expensive and laborious to maintain the maps [61]. Several works have been proposed to learn the road graph topology such as [62, 63]. Besides the topology of map, detailed features of HD map are also required for robust sensing and localization. Among all the features of HD maps, semantic attributes are crucial for accurate and robust mapping, providing an abstraction from the raw

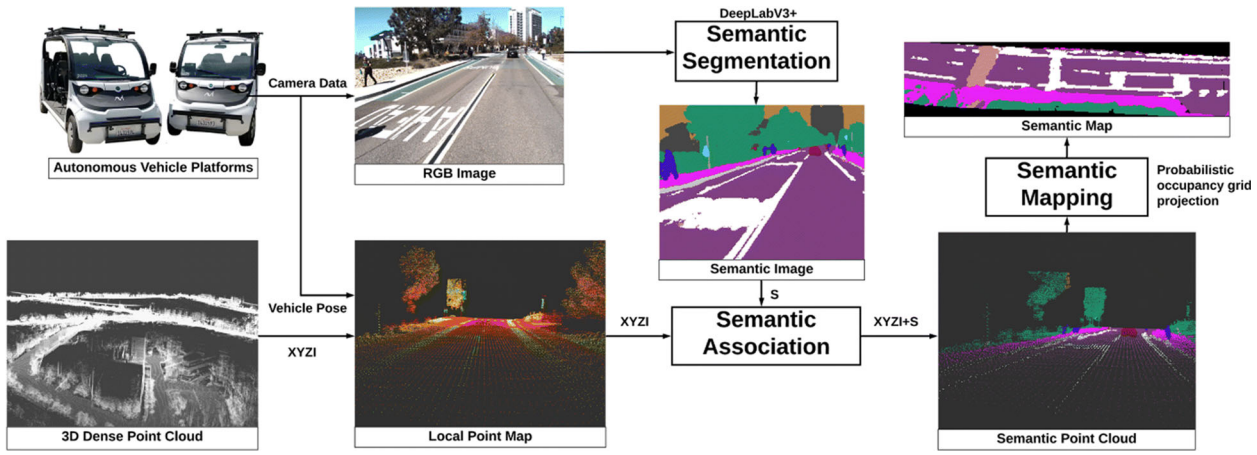


Fig. 32 Probabilistic Semantic Mapping pipeline

image data and abundant details about the static environment. To tackle this problem, several methods have been proposed to automate/semi-automate this process. In [64], a hierarchical Conditional Random Fields approach was combined with pairwise potential minimization. Sengupta et al. [65] incorporate stereo pairs for depth image estimation. Westfechtel et al. [66] used LiDAR-based SLAM to build a large scale 3D semantic map. Random forest and deep learning methods were also applied to reclassify the label.

Incorporating the road priors can benefit more robust semantic mapping and provides a better way of capturing the uncertainties. Inspired by this, we have proposed a probabilistic semantic mapping approach [61]. Our pipeline for building the semantic maps is shown in Fig. 32. We first use DeepLabV3Plus [67] to do 2D semantic segmentation and associate semantic pixels with the corresponding 3D LiDAR points by using the extrinsics between camera and LiDAR. After the semantic association step, we use the proposed semantic mapping algorithm to update the bird's eye view (BEV) semantic belief by using confusion matrix and LiDAR intensity prior. The confusion matrix is utilized to estimate the uncertainty of the semantic segmentation networks, while the LiDAR intensity is used to capture the infrastructure prior as the road elements have different reflectivities. The update rule for semantic mapping follows the classical filtering algorithm with additional consideration of the above priors. For details of the algorithm, please refer to the probabilistic semantic mapping paper [61].

We tested the above algorithm using our campus environment with several challenging scenarios including junctions, hills, and construction sites. The generated global map using our best model is shown in Fig. 33.

Quantitative results are shown in Table 4. The Vanilla approach does not consider any mentioned priors and only updates the belief with uniform probability. Vanilla+I considers the LiDAR intensity prior. CFN uses the confusion matrix as the probability to update the belief. Finally, CFN+I considers, both the intensity and the confusion matrix. As we can see, adding LiDAR intensity can improve baseline's accuracy and IoU of lane marks as the intensity is a strong indicator of potential lane marks. Also, the confusion matrix will benefit the estimation of crosswalks and lane marks by a large margin. In the next section, we explore the applications of probabilistic semantic maps for dynamic trajectory generation in urban environments to explore alternative to HD maps.

4.4 Dynamic trajectory generation (TridentNet)

Classical global planner, behavior planner and local planner pipeline usually has high dependency on HD maps

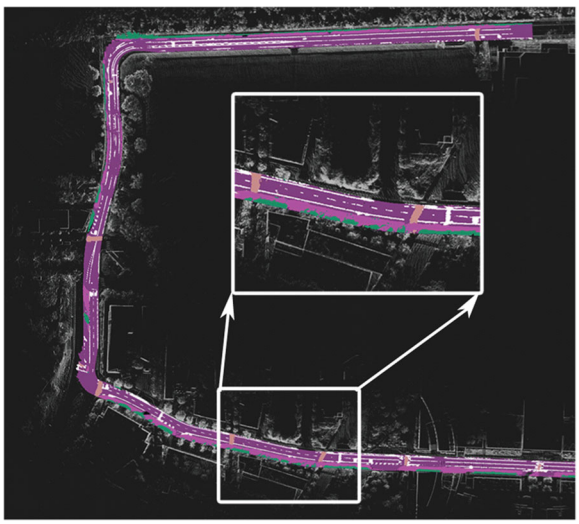


Fig. 33 The BEV of generated semantic map of a campus environment

Table 4 Quantitative evaluation of probabilistic semantic mapping algorithm. RD, CW, LM refer to roads, crosswalks, lane marks respectively

Methods	IoU			mIoU	Accuracy		
	RD	CW	LM		RD	CW	LM
Vanilla	0.715	0.537	0.135	0.462	0.797	0.577	0.180
Vanilla+I	0.712	0.510	0.163	0.462	0.789	0.548	0.234
CFN	0.671	0.605	**0.301**	0.526	0.708	0.696	0.652
CFN+I	0.669	0.588	0.298	0.518	0.705	0.676	0.657

which have poor scalability as the environment is constantly evolving due to events such as construction and road closure. As we have discussed in the previous section, maintaining the HD map is expensive and laborious. Thus, reducing the dependency of detailed map is required for more robust navigation and planning. Moreover, lowering the barrier to entry by making the technological requirements more accessible is essential to the advancement of autonomous driving, as the benefits are dependent on widespread adoption of the technology. To tackle the challenges, several end-to-end learning based methods have been proposed to learn control actions based on coarse directional cues and current perceived environment. Hecker et al. [68] used a coarse map combined with LSTM to estimate steering and speed controls. Amini et al. [69] built a variational network to learn the point-to-point navigation and localization based on noisy GPS data and coarse maps. However, as the control actions are coupled with vehicle-specific configurations, the approach may not scale and apply to other vehicle platforms. To decouple path planning from control, we focus on dynamically generating trajectories that can be used in combination with existing path tracking and kinematics models as described in Section 2.2.4. Given that the trajectories may be multi-modal as there exist several paths to reach the destination given the goal and dynamic environment, we propose a conditional generative model called TridentNet [70] to generate dynamic trajectories utilizing only a global plan based on coarse OpenStreetMap (OSM) [71] and a local probabilistic semantic map as described in Section 4.3.

The proposed pipeline is shown in Fig. 34. Instead of using detailed HD maps to generate the navigation waypoints, a coarse OSM is used to provide high-level directional cues. More specifically, our global planner implementation will output the shortest path from the current pose to the destination by using only the road graph of OSM. We then use a raster mask to embed those directional cues. Since the OSM may not be able to capture complete contextual information including road markings, the probabilistic map is used; this map can be updated with the semantic belief as car perceives new observations of the environment [61]. Convolutional Neural Networks are used to learn feature embeddings of rasterized directional cues and local probabilistic maps. Afterwards, a Conditional Variational Autoencoder (CVAE) [72] is utilized to generate multiple trajectory hypotheses. The latent variables of the CVAE are capable of capturing different driving modes given the directional representations. During training, multiple latent variables are sampled, and during inference, the latent variable with the highest confidence score is used to generate the planned trajectory. This trajectory can then be processed by the downstream path tracking algorithm and controller. Additional details about the methods can be found in TridentNet [70].

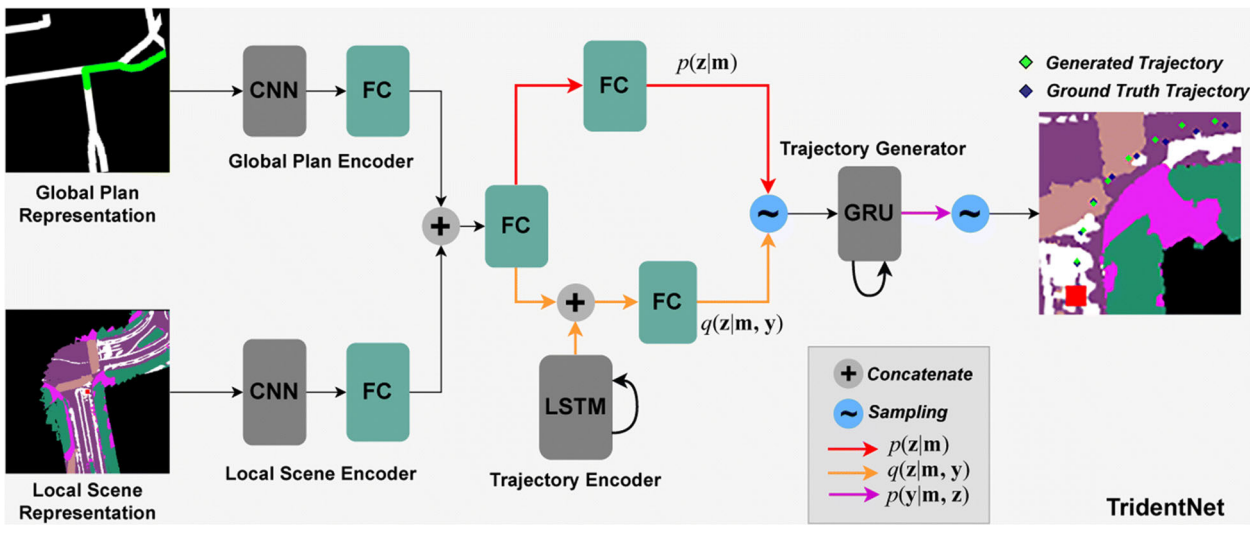


Fig. 34 Architecture of TridentNet

Table 5 Evaluation results of TridentNet on collected UCSD dataset

UC San Diego Dataset				
Model	ADE_{FULL}	ADE_{HALF}	FDE	MDE
TridentNet-**H**10-**S**3	**1.056245**	**0.336941**	**2.447714**	**2.494614**
TridentNet-**H**15-**S**2	2.341875	0.753802	6.127183	6.177646

To evaluate the performance, we adopt Average Displacement Error (ADE) and Final Displacement Error (FDE), which is commonly used in trajectory prediction literature [73]. Additionally, Maximum Displacement Error (MDE) is used to measure the worst waypoint predicted along a trajectory. The results for two different models are shown in Table 5. The first model, TridentNet-**H**10-**S**3, is configured to dynamically generate trajectories that are represented by 10 waypoints spaced $3m$ apart; on the other hand, TridentNet-**H**15-**S**2 uses 15 waypoints spaced by $2m$ apart. As the initial set of waypoints are more likely to be executed by the ego-vehicle, those waypoints are critical for robust and safe navigation. To this end, ADE_{HALF} is used to evaluate short-term performance. While both of the models can generate trajectories up to $30m$, **H**10-**S**3 decodes less waypoints compared to **H**15-**S**2, and as a result reduces potential compound errors. Test samples generated from **H**10-**S**3 can be seen in Fig. 35.

Our proposed models show low relative error in terms of ADE_{HALF}. Thus, indicating their potential capabilities of capturing short-term intent and generating accurate planned paths while eliminating dependencies on manually labeled HD maps. Additionally, our results present future research directions on obstacle avoidance, motion planning, and closing the gap with perception modules such as prediction.

4.5 Building a digital twin

Testing and verification for systems that are intended to operate in safety critical scenarios is indispensable. During the development process, our team used a combination of fully simulated modules and data-driven techniques to verify localization, perception, and planning strategies. This provided early indication on how the various components would perform separately and jointly. However, in various cases, it was determined that fully simulated data did not provide a comprehensive representation of real scenarios. While it is possible to collect new data with a full-scale vehicle, this presents a considerable overhead and may not scale if manual annotation is involved, or a larger number of related test samples are needed. This motivates the direction for building a digital model for simulation that not only provides the convenience of fully simulated methods but additionally provides representative data that matches real-world testing. In this section, we discuss efforts to evaluate larger scale data collection methods, as a means to create synthetic models representative of the real world, for algorithmic testing.

4.5.1 From world to world model

Map creation for autonomous driving highlights the need for temporal model representations of the world to be sufficiently close to the real-world to ensure safe navigation. Rapidly changing road conditions, from construction works to lane re-painting and vegetation growth on nearby embankments, all emphasize the requirement for scalable map generation techniques. Google Streetview [74] was one of the first efforts to leverage road vehicle mounted sensors to acquire map data at scale. With the requirements of fleets of vehicles to map every street across nations, the complexity and delays involved to track changes have made Google Maps a best-effort solution to resolve map creation. Alternative efforts, such as those to crowd-source sensor data [75], have emphasized

Fig. 35 Test samples generated by **H**10-**S**3 are shown for lane following (a), an intersection right-turn (b), and an intersection left-turn (c). Ground truth waypoints shown in blue and predicted waypoints are shown in green

the pain-points associated with geo-referencing and data requirements of data collection. We believe that HD Maps are unsustainable at scale, but that high-level road information may be a reasonable prior, paired with dynamic trajectory generation, as discussed in Section 4.4. As such, we investigate the options of using aerial data for map and model creation.

On the UC San Diego campus, we tackled scalability of map data collection across a campus-wide effort by using Unmanned Aerial Vehicles (UAV) in conjunction to vehicle mounted sensors. Through RTK-GPS referenced aerial image collection with a Phantom 4-RTK platform, a collection of over 1,400 images across 2 flights resulted in 0.38 km^2 coverage with a Ground Sampling Distance (GSD) of 1.8cm/pixel, shown in Figs. 36 and 37. All data georeferencing was completed using aerial pose estimates derived from RTK based timing correction to GPS to achieve <2cm localization accuracy in a global coordinate system. While the data collected was a progressive increment towards larger area scanning, it was concluded that aerial-based map data collection efforts provide a valid and feasible path towards state and nationwide timely reference map creation.

The benefits of efficient, large-area data acquisitions did not come without difficulties. Firstly, bringing multi-altitude UAV and satellite data together resulted in feature discontinuities such as lane markings. Geo-referencing with state-of-art RTK systems provided localization accuracies to sub 5-cm for UAV data, which was still too large to ensure perfect alignment. It was also found that many public map contents also adhered to georeferencing standards on the order of 50cm, as observed in Fig. 38. Additionally, dense multi-view reconstruction techniques for image based reconstructions commonly use depth filtering to reduce reconstruction noise. Artifacts such as power-lines, poles and traffic signs were all subject to occasional filtering, hence being removed from the reconstructions. While there are methods to handle thin-features in multi-view reconstructions [76], these are often missing in open-source reconstruction libraries. The findings highlighted important considerations when attempting to use only real-world aerial data for map creation in the scope of autonomous vehicles. Table 6 summarizes the approximate spatial resolutions and geo-referencing capabilities associated with different scales of data acquisition systems including UAV [77], aerial and satellite maps [78] as well as ground vehicle [79] based map creation systems. While some overlap, it can be seen that manned aircraft with high spatial imaging resolution systems offers both high spatial density (GSD) as well as strong geo-referencing, with the benefit of being even more scalable than UAV based mapping. Future improvements in sensory packages and platforms will yield further options for scalable, real-world data capture methods that are applicable to map creation for autonomous ground vehicles.

Fig. 36 UC San Diego engineering campus 3D point cloud from geo-referenced UAV imaging

4.5.2 Synthetic driven data

The lack of readily available ground-truth data from sensory perspectives in real-world data encourages the creation of fully synthetic data. Such synthetic models can be real-world based (derived from captured data but fitted to pre-defined geometric primitives) or designed fictionally. As a step towards this end, various autonomous driving simulators were explored including CARLA [80] and LGSVL [81]. These provide tools and methodologies for close loop testing with existing autonomous driving frameworks that include map customization tools, physics engines, sensor models, as well as simulated road users. Although both simulators provide highly customizable toolkits for map generation, CARLA incorporates direct API support for the RoadRunner mapping software which facilitates georeferenced road network mapping in addition to a suite of configurable sensor types as shown in Fig. 39. RoadRunner utilizes a combination of aerial imagery, elevation maps, and point clouds to generate road network definitions that account for various geometries and elevation characteristics. Leveraging the UAV data captured and publically available satellite imagery, a digital twin was initially generated with complete road networks for approximately 70% of the UC San Diego campus; three parts of campus are shown in Figs. 40(a), 40(c), and 40(e). All data was referenced to a local Cartesian coordinate system with a global offset.

Although the initial road network definition for campus can be applied for verification across planning strategies given the ROS bridge support for our navigation stack, the gap between the simulated perception and real sensor data persists. Additionally, generating realistic simulation scenarios relies on accurate representation of road furniture, buildings, vegetation, traffic signs, etc. Having

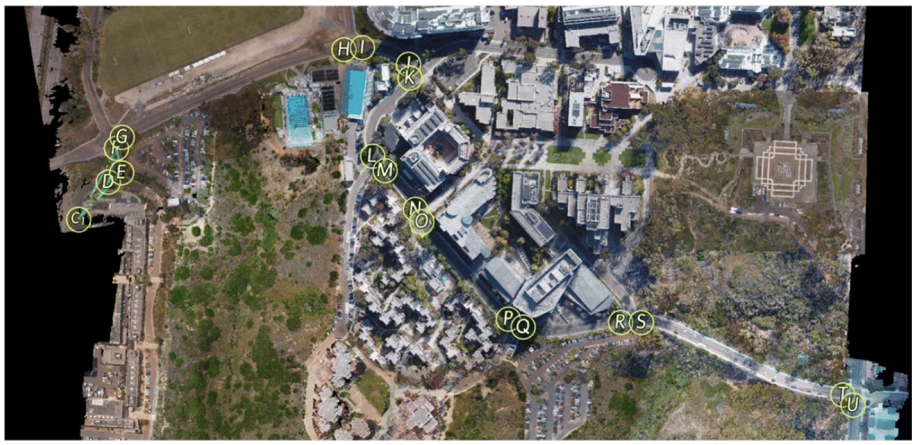


Fig. 37 UC San Diego engineering campus 2D orthogonal view with marked stop signs from geo-referenced UAV imaging

developed campus wide semantic models as described in Section 4.3 and experimented with semantic scene models for dynamic trajectory generation to address scalability limitations (Section 4.4), we first explore a semantic representation strategy. This representation is portrayed in Figs. 40(b), 40(d), and 40(f) following the color map from our work on semantic mapping. Given that our approach for dynamic trajectory generation operates with semantic representations, this presents an opportunity for testing and verification, but also in terms of generating unseen scenarios during navigation that can be useful during training machine learning based models. Secondly, for use-cases the depend on photorealistic representations, as shown in Figs. 36 and 37, an active research direction involves incorporating photorealistic 3D models through the process of surface reconstruction.

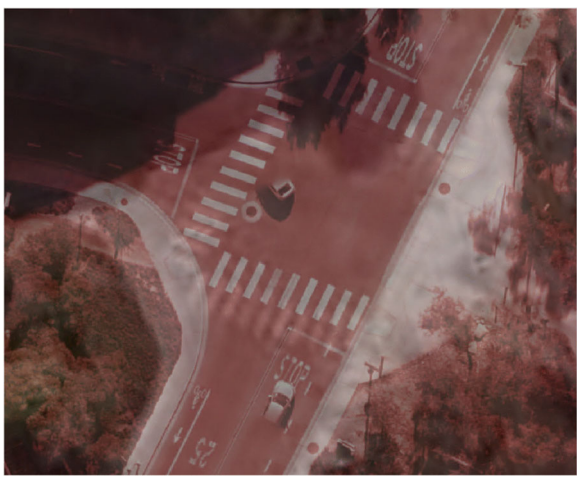


Fig. 38 Semi-transparent overlay of UAV captured, geo-referenced orthomosaic image data over Google satellite imagery of intersection at the UC San Diego engineering campus. Inconsistencies of pedestrian crossings emphasize the issues associated with geo-referencing, and temporal data agreement, in addition to strong scale differences

5 Lessons and issues for the future

The design of the vehicles for urban mobility has made it clear that there are still a number of challenges for the future. The challenges span the full system architecture.

Driving in an area with a lot of pedestrians requires the ability to detect other road-users and predict their intent. The present system is good at detecting pedestrians, but the prediction of future motion is at best 1-3 seconds, which is not enough to safely avoid a collision if a person steps onto the road.

A similar problem is experienced with cars. The vision and LiDAR systems are only line of sight. In traffic, the system only sees the car immediately in front of it. Consequently, we are slowly evaluating use of radar systems from several providers to determine the best way to improve situational awareness. The radars also have operating long-range, which is needed to be able to detect other cars at intersections. Entering a busy street from a side street requires detection of cars at least 450 ft / 150 m away. In addition, longer range detection and tracking is desirable to allow for robust intent recognition over a period of 10+ seconds.

An interesting issue that has not yet been fully studied in the system architecture includes both, hardware and

Table 6 Approximate real-world geospatial map data source comparisons for GSD and geo-referencing accuracies

Data Sources	GSD (cm/pix)	Geo-Referencing (cm)
UAV	1-10	5
Satellite	20-100	50+
Manned Aircraft	5-20	10
Ground Vehicle	1-10	5

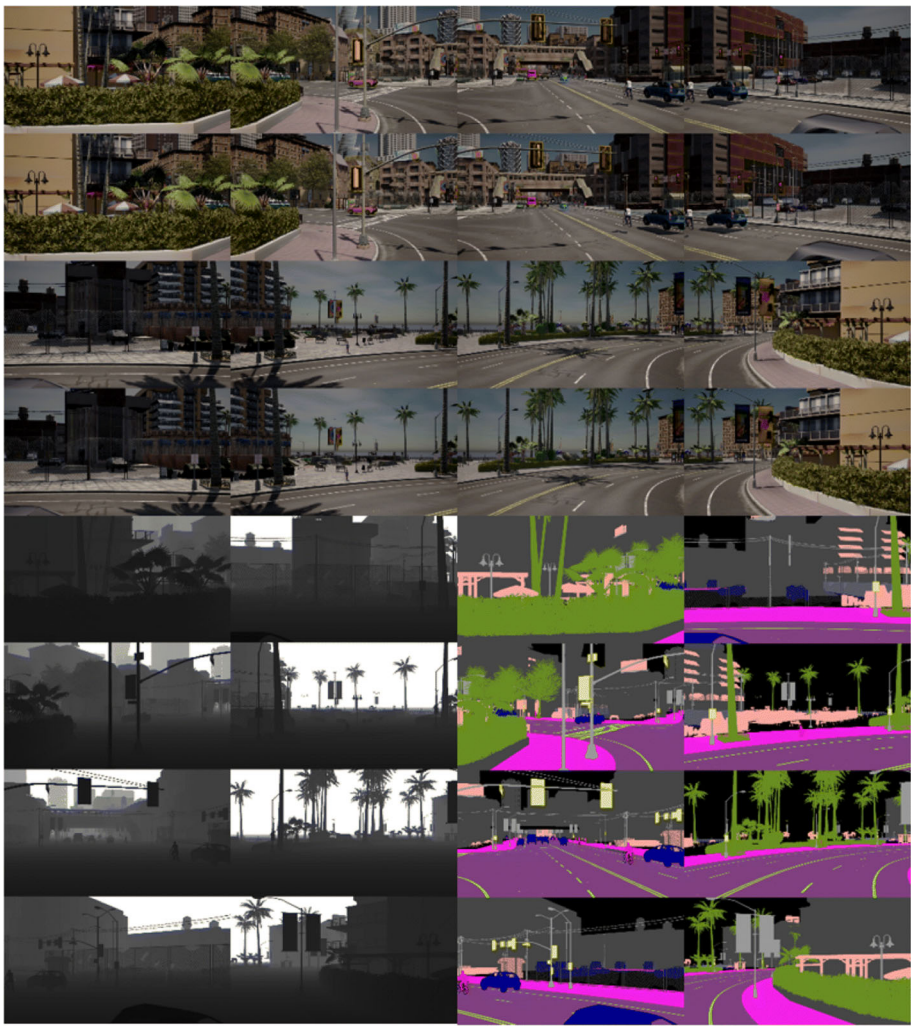


Fig. 39 Synthetic stereo panoramic camera data generated with CARLA. From top to bottom: RGB images of cameras, bottom-left: respective depth maps, bottom-right: respective semantic segmentation maps

software components. Recently, a proliferation of high-performance embedded systems have entered the market such as the NVIDIA Jetson Nano and the Qualcomm RB5 system. The systems can interface to 4+ cameras, and have embedded GPUs. The systems can perform 10+ trillion operations per second. As such, an interesting question is how to partition the system to have much of the processing carried out at the edge. One could imagine a system that is a federation of embedded systems interconnected by high-speed Ethernet, which would reduce the need for cabling and allow for much higher flexibility, but debugging such systems is much harder. Finding the right balance is an open research challenge.

The role of maps is another interesting question. Most AV companies rely on HD-maps, which poses a challenge in terms of updating and maintenance. We have adopted a more minimalistic approach with a hierarchy of map representations of high-level topological graphical models (such as Open Street Maps) over local maps to instantaneous models that are updated in real-time. At the same time there is a need for a semantic dimension as well to characterize the relative location of sidewalks, traffic signs, crosswalks, ... The semantic models provide a strong prior on the interpretation of the environment which in term allow for prediction of the layout of the environment and the behavior / intent of other road-users. Finding the right balance between these prior and online map representations is an interesting research question in its own right.

The value of a digital twin is another issue that we have only started to explore. There are numerous use-cases for such a system. It is possible to simulate the system and explore many issues for behavior verification, for training of ML models to bootstrap models prior to real-world

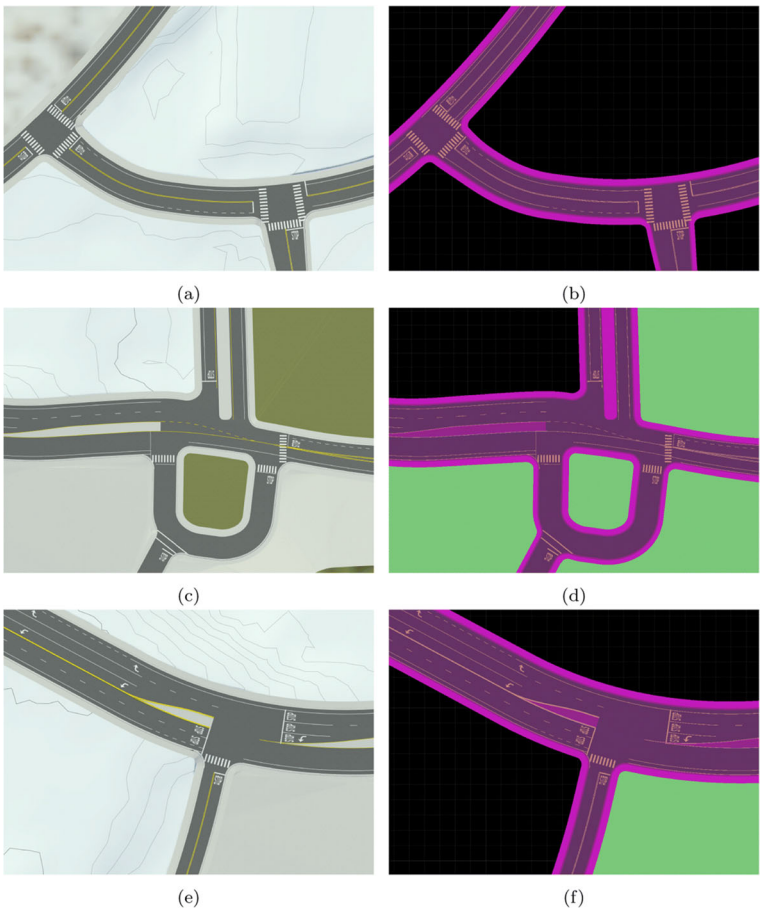


Fig. 40 Various BEV road network definitions for the UC San Diego campus. On the left column ((a)(c) and (e)), the road networks are generated using standard road textures provided by RoadRunner. The corresponding semantic maps are shown in (b)(d), and (f)

fine-tuning. The twin system can also be used for post-mission analysis of system behavior. As we use the ROS system, the default data-viewer is RViz, which is great, but it lacks a lot of the tools for analysis. In addition, the digital twin is interesting for level 4-5 systems as it is possible to augment the digital twin system with an AR/VR embedding that would allow remote supervision and tele-operation, which has numerous advantages for training and monitoring of systems during deployment. Obviously, the remote monitoring poses new challenges in terms of adequate / robust communication links to ensure safe operation and the design of robust control methods in the presence of variable time-delays.

Finally, there is a need to consider new techniques for software engineering. The present design with a mixture of Python and C++/ROS is very flexible but it is difficult to have a robust framework for debugging and to provide any form of safety guarantee. Design of mass scale autonomous systems will require mature software tools with an ability to design, simulate, code-generate and debug systems at multiple levels of abstraction. We are far from such a system, and it is very much a research challenge for the future.

The effort at UC San Diego in the Autonomous Vehicle Laboratory is an interesting start, and we have made serious progress both in terms of systems realization and in terms of long-term evaluation. At the same time, we are just getting started. There are many more M.Sc. and Ph.D. theses to be written before we are at the end. In parallel, we are trying to transition some of our results to industry to demonstrate impact beyond the academic environment.

Acknowledgements
We acknowledge the support from Dr. Todd Hylton and members of the Autonomous Vehicle Lab at UC San Diego that have shared valuable input throughout the course of our research, specifically co-authors in [10, 25, 61]. We thank Prof. Falko Kuester, Eric Lo and Jonathan Klingspon, at the UC San Diego Dronelab, for providing access to the data in Section. 4.5. We also appreciate the partnership and collaboration with UC San Diego facilities, mailing center, and police station for supporting our research.

Authors' contributions
Dr. Henrik I. Christensen directed the project described in this manuscript. David Paz' main contributions include Sections 2, 3, 4.3, and 4.4; additional efforts and input were provided in Sections 4.1, and 4.5. Hengyuan Zhang's

main contributions include Sections 2.2, 4.1, 4.2, and 4.3; additional efforts and input were provided in Sections 2, 3, and 4.4. Dominique Meyer contributed significantly to Section 4.5; he also provided valuable input in Sections 2, 3. Hao Xiang's main contributions include Section 4.3; he has also provided valuable input to Section 4.4. Yunhai Han and Yuhan Liu contributed significantly to Section 4.1. Andrew Liang, Zheng Zhong, and Shiqi Tang's main contributions include Sections 2.3, 3.2.2, and 4.5, respectively. The authors read and approved the final manuscript.

Funding
The effort has been funded by UC San Diego through the Contextual Robotics Institute with support from Qualcomm.

Availability of data and materials
The datasets associated with Sections 4.1.1 and 4.4 and related search efforts can be accessed from http://avl.ucsd.edu.

Code availability
Various code repositories from this work can be found at https://github.com/AutonomousVehicleLaboratory.

Declarations

Competing interests
The authors do not have any external engagement related to this project.

Received: 3 August 2021 Accepted: 14 September 2021
Published online: 22 November 2021

References

1. E. D. Dickmanns, A. Zapp, in *Mobile Robots, SPIE*, ed. by Wolfe W.J., Marquina N. A Curvature-based Scheme for Improving Road Vehicle Guidance by Computer Vision, vol. 727, (Bellingham, 1987), pp. 161–168
2. E. Dickmanns, *Dynamic Vision for Perception and Control of Motion*. (Springer Verlag, Heidelberg, 2007)
3. M. A. Turk, D. G. Morgenthaler, K. D. Gremban, M. Marra, VITS - A Vision System for Autonomous Land Vehicle Navigation. IEEE Trans. Pattern Anal. Mach. Intell. **10**(3), 342–361 (1988)
4. B. Marr, Key milestones of Waymo - Google's self-driving cars (2018). https://forbes.com/sites/bernardmarr/2018/09/21/key-milestones-of-waymo-googles-self-driving-cars. Accessed 02 Oct 2021
5. I. Bonifacic, Toyota is developing autonomous taxis with help from Aurora (2021). Engadget. https://www.engadget.com/toyota-aurora-denso-autonomous-vehicle-partnership-191500404.html. Accessed 02 Oct 2021
6. A. Palmer, Amazon Zoox unveils self-driving robotaxi (2020). CNBC, https://cnbc.com/2020/12/14/amazons-self-driving-company-zoox-unveils-autonomous-robotaxi.html. Accessed 02 Oct 2021
7. A. Adler, Aurora closes in on production version of self-driving truck technology (2021). FreightWaves, https://www.freightwaves.com/?p=358829. Accessed 02 Oct 2021
8. Baidu-Apollo-team, Apollo: Open source autonomous driving (2017). https://github.com/ApolloAuto/apollo. Accessed 02 Oct 2021
9. S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, T. Hamada, An open approach to autonomous vehicles. IEEE Micro. **35**(6), 60–68 (2015). https://doi.org/10.1109/MM.2015.133
10. D. Paz, P.-J. Lai, S. Harish, H. Zhang, N. Chan, C. Hu, S. Binnani, H. Christensen, in *Field and Service Robotics*. Lessons learned from deploying autonomous vehicles at UC San Diego, (Tokyo, JP, 2019)
11. Z. Zhang, A flexible new technique for camera calibration. IEEE Trans. Pattern Anal. Mach. Intell. **22**(11), 1330–1334 (2000)
12. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Ng, in *ICRA Workshop on Open Source Software*. ROS: an open-source Robot Operating System, vol. 3, (2009)
13. A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, O. Beijbom, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. PointPillars: Fast encoders for object detection from point clouds, (2019), pp. 12689–12697. https://doi.org/10.1109/CVPR.2019.01298
14. M. Fischler, R. Bolles, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM. **24**, 381–395 (1981)
15. V. Lepetit, F. Moreno-Noguer, P. Fua, EPnP: An accurate O(n) solution to the PnP problem. Int. J. Comput. Vis. **81**, 155 (2009). https://doi.org/10.1007/s11263-008-0152-6
16. J. Redmon, A. Farhadi, YOLOv3: An Incremental Improvement. ArXiv abs/1804.02767 (2018)
17. C. Campos, R. Elvira, J. Rodríguez, J. Montiel, J. D. Tardós, ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM. ArXiv abs/2007.11898 (2020)
18. J. Zhang, S. Singh, in *Robotics: Science and Systems*. LOAM: Lidar odometry and mapping in real-time, (2014)
19. M. Magnusson, The three-dimensional normal-distributions transform — an efficient representation for registration, surface analysis, and loop detection. PhD dissertation, Örebro universitet (2009)
20. R. B. Rusu, S. Cousins, in *IEEE International Conference on Robotics and Automation (ICRA)*. 3D is here: Point Cloud Library (PCL), (Shanghai, China, 2011)
21. H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, O. Beijbom, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. nuScenes: A multimodal dataset for autonomous driving, (2020). https://doi.org/10.1109/CVPR42600.2020.01164
22. C. Ming-Fang, L. John, S. Patsorn, S. Jagjeet, B. Slawomir, H. Andrew, D. Wang, C. Peter, L. Simon, R. Deva, H. James, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Argoverse: 3D tracking and forecasting with rich maps, (2019), pp. 8740–8749
23. H. Darweesh, E. Takeuchi, K. Takeda, Y. Ninomiya, A. Sujiwo, Y. Morales, N. Akai, T. Tomizawa, S. Kato, Open source integrated planner for autonomous navigation in highly dynamic environments. J. Robot. Mechatron. **29**, 668–684 (2017). https://doi.org/10.20965/jrm.2017.p0668
24. R. C. Coulter, Implementation of the pure pursuit path tracking algorithm. Tech. rep. Carnegie-Mellon UNIV Pittsburgh PA Robotics INST (1992)
25. D. Paz, P.-J. Lai, N. Chan, Y. Jianf, H. I. Christensen, in *International Conference on Intelligent Robots and Systems (IROS)*. Autonomous vehicle benchmarking using unbiased metrics (IEEE/RSJ, Las Vegas, NV, 2020a)
26. A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, A. Alahi, 2018, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks, pp. 2255–2264. https://doi.org/10.1109/CVPR.2018.00240
27. T. Fernando, S. Denman, S. Sridharan, C. Fookes, in *Computer Vision – ACCV 2018*, ed. by C. V. Jawahar, H. Li, G. Mori, and K. Schindler. GD-GAN: Generative Adversarial Networks for Trajectory Prediction and Group Detection in Crowds (Springer International Publishing, Cham, 2019), pp. 314–330
28. J. Amirian, J. Hayet, Pettré J, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Social Ways: Learning multi-modal distributions of pedestrian trajectories with GANs, (2019), pp. 2964–2972
29. A. Sadeghian, V. Kosaraju, A. Sadeghian, N. Hirose, H. Rezatofighi, S. Savarese, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. SoPhie: An attentive GAN for predicting paths compliant to social and physical constraints, (2019), pp. 1349–1358. https://doi.org/10.1109/CVPR.2019.00144
30. V. Kosaraju, A. Sadeghian, R. Martín-Martín, I. Reid, H. Rezatofighi, S. Savarese, in *Advances in Neural Information Processing Systems, Curran Associates, Inc.*, ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett. Social-bigat: Multimodal trajectory forecasting using bicycle-gan and graph attention networks, vol. 32, (2019). https://proceedings.neurips.cc/paper/2019/file/d09bf41544a3365a46c9077ebb5e35c3-Paper.pdf
31. T. Salzmann, B. Ivanovic, P. Chakravarty, M. Pavone, Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data (2020). https://arxiv.org/pdf/2001.03093.pdf
32. X. Feng, Z. Cen, J. Hu, Y. Zhang, in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. Vehicle trajectory prediction using intention-based conditional variational autoencoder, (2019), pp. 3514–3519. https://doi.org/10.1109/ITSC.2019.8917482
33. A. Mohamed, K. Qian, M. Elhoseiny, C. Claudel, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Social-STGCNN: A social spatio-temporal graph convolutional neural network for human trajectory prediction, (2020), pp. 14412–14420. https://doi.org/10.1109/CVPR42600.2020.01443

34. J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, C. Schmid, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. VectorNet: Encoding hd maps and agent dynamics from vectorized representation, (2020), pp. 11522–11530. https://doi.org/10.1109/CVPR42600.2020.01154
35. A. Ribeiro, L. Dihl, C. Jung, in *International Conference on Systems, Signals and Image Processing*. Automatic camera calibration for driver assistance systems, (2006), pp. 173–176
36. L. Lu, X. Lu, S. Ji, C. Tong, in *Intelligent Information Processing VII*. A traffic camera calibration method based on multi-rectangle (Springer, Berlin, Heidelberg, 2014), pp. 230–238
37. H. Wang, Y. Cai, G. Lin, W. Zhang, A novel method for camera external parameters online calibration using dotted road line. Adv. Robot. **28**, 1033–1042 (2014). https://doi.org/10.1080/01X00000.6918642014.902329
38. A. Geiger, P. Lenz, C. Stiller, R. Urtasun, Vision meets robotics: The KITTI dataset. Int. J. Robot. Res. (IJRR). **32**(11), 1231–1237 (2013)
39. Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, R. Girshick, Detectron2 (2019). https://github.com/facebookresearch/detectron2. Accessed 02 Oct 2019
40. C. B. Madsen, H. I. Christensen, *Chapter 1. Modelling and testing the stability of edge segments: Length and orientation*. (World Scientific Press, Singapore, 1995), pp. 1–15
41. K. He, G. Gkioxari, P. Dollár, R. Girshick, in *2017 IEEE International Conference on Computer Vision (ICCV)*. Mask R-CNN, (2017), pp. 2980–2988. https://doi.org/10.1109/ICCV.2017.322
42. R. Grompone von Gioi, G. Randall, A Sub-Pixel Edge Detector: an Implementation of the Canny/Devernay Algorithm. Image Process. On Line. **7**, 347–372 (2017). https://doi.org/10.5201/ipol.2017.216
43. Y. Han, Y. Liu, D. Paz, H. Christensen, Auto-calibration method using stop signs for urban autonomous driving applications. ArXiv abs/2010.07441 (2021). https://doi.org/2010.07441
44. C. Park, P. Moghadam, S. Kim, S. Sridharan, C. Fookes, Spatiotemporal Camera-LiDAR calibration: A targetless and structureless approach. IEEE Robot. Autom. Lett. **5**, 1556–1563 (2020)
45. J. Kümmerle, T. Kühner, in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. Unified intrinsic and extrinsic camera and LiDAR calibration under uncertainties, (2020), pp. 6028–6034. https://doi.org/10.1109/ICRA40945.2020.9197496
46. A. Dhall, K. Chelani, V. Radhakrishnan, K. M. Krishna, LiDAR-Camera Calibration using 3D-3D Point correspondences. ArXiv e-prints 1705.09785 (2017)
47. S. Mishra, G. Pandey, S. Saripalli, in *2020 IEEE Intelligent Vehicles Symposium (IV)*. Extrinsic calibration of a 3D-LIDAR and a camera, (2020), pp. 1765–1770
48. P. Torr, A. Zisserman, MLESAC: A new robust estimator with application to estimating image geometry. Comput. Vis. Image Underst. **78**, 138–156 (2000)
49. R. B. Rusu, Semantic 3D object maps for everyday manipulation in human living environments. PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany (2009)
50. T. Yin, X. Zhou, P. Krähenbühl, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Center-based 3D object detection and tracking, (2021), pp. 11784–11793
51. B. Zhu, Z. Jiang, X. Zhou, Z. Li, G. Yu, Class-balanced grouping and sampling for point cloud 3D object detection. ArXiv abs/1908.09492 (2019)
52. J. V. Brummelen, M. O'Brien, D. Gruyer, Najjaran H, Autonomous vehicle perception: The technology of today and tomorrow. Transp. Res. C Emerg. Technol. **89**, 384–406 (2018). https://doi.org/10.1016/j.trc.2018.02.012
53. J. Elfring, R. Appeldoorn, S. Dries, M. Kwakkernaat, Effective world modeling: Multisensor data fusion methodology for automated driving. Sensors (Basel, Switzerland). **16**, 1668 (2016)
54. A. Rangesh, M. M. Trivedi, No blind spots: Full-surround multi-object tracking for autonomous vehicles using cameras and lidars. IEEE Trans. Intell. Veh. **4**(4), 588–599 (2019). https://doi.org/10.1109/TIV.2019.2938110
55. Z. Ding, Y. Hu, R. Ge, L. Huang, S. Chen, Y. Wang, J. Liao, 1st place solution for Waymo open dataset challenge - 3D detection and domain adaptation. ArXiv abs/2006.15505 (2020)
56. H. Cho, Y. Seo, B. V. K. V. Kumar, R. R. Rajkumar, in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. A multi-sensor fusion system for moving object detection and tracking in urban driving environments, (2014), pp. 1836–1843. https://doi.org/10.1109/ICRA.2014.6907100
57. C. R. Qi, W. Liu, C. Wu, H. Su, L. J. Guibas, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Frustum PointNets for 3D object detection from RGB-D data, (2018), pp. 918–927. https://doi.org/10.1109/CVPR.2018.00102
58. D. Xu, D. Anguelov, A. Jain, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Pointfusion: Deep sensor fusion for 3D bounding box estimation, (2018), pp. 244–253
59. X. Weng, J. Wang, D. Held, K. Kitani, *3D Multi-Object Tracking: A Baseline and New Evaluation Metrics*, (2020), pp. 10359–10366
60. H.-k. Chiu, A. Prioletti, J. Li, J. Bohg, Probabilistic 3d multi-object tracking for autonomous driving. ArXiv abs/2001.05673 (2020)
61. D. Paz, P. J. Lai, N. Chan, Y. Jianf, H. I. Christensen, *Probabilistic semantic mapping for urban autonomous driving applications* (IEEE, 2020), pp. 2059–2064
62. N. Homayounfar, W. C. Ma, J. Liang, X. Wu, J. Fan, R. Urtasun, in *Proceedings of the IEEE/CVF International Conference on Computer Vision*. Dagmapper: Learning to map by discovering lane topology, (2019), pp. 2911–2920
63. N. Homayounfar, W.-C. Ma, S. K. Lakshmikanth, R. Urtasun, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Hierarchical recurrent attention networks for structured online maps, (2018), pp. 3417–3426
64. S. Sengupta, P. Sturgess, L. Ladickỳ, P. H. Torr, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Automatic dense visual semantic mapping from street-level imagery (IEEE, 2012), pp. 857–862
65. S. Sengupta, E. Greveson, A. Shahrokni, P. H. Torr, in *2013 IEEE International Conference on robotics and Automation*. Urban 3D semantic modelling using stereo vision (IEEE, 2013), pp. 580–585
66. T. Westfechtel, K. Ohno, R. P. B. Neto, S. Kojima, S. Tadokoro, in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. Fusion of camera and lidar data for large scale semantic mapping (IEEE, 2019), pp. 257–264
67. L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, H. Adam, in *Computer Vision - ECCV 2018. Lecture Notes in Computer Science*, ed. by V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss. Encoder-decoder with atrous separable convolution for semantic image segmentation, vol. 11211 (Springer, 2018)
68. S. Hecker, D. Dai, L. Van Gool, in *Proceedings of the european conference on computer vision (ECCV)*. End-to-end learning of driving models with surround-view cameras and route planners, (2018), pp. 435–453
69. A. Amini, G. Rosman, S. Karaman, D. Rus, in *2019 International Conference on Robotics and Automation (ICRA)*. Variational end-to-end navigation and localization (IEEE, 2019), pp. 8958–8964
70. D. Paz, H. Zhang, H. I. Christensen, *TridentNet: A conditional generative model for dynamic trajectory generation*, (Singapore, 2021)
71. M. Haklay, P. Weber, Openstreetmap: User-generated street maps. IEEE Pervasive Comput. **7**(4), 12–18 (2008)
72. K. Sohn, X. Yan, H. Lee, in *Proceedings of the 28th International Conference on Neural Information Processing Systems*. Learning structured output representation using deep conditional generative models, vol. 2 (MIT Press, Cambridge, 2015), pp. 3483–3491
73. T. Yang, Z. Nan, H. Zhang, S. Chen, N. Zheng, in *2020 IEEE Intelligent Vehicles Symposium (IV)*. Traffic agent trajectory prediction using social convolution and attention mechanism (IEEE, 2020), pp. 278–283
74. D. Anguelov, C. Dulong, D. Filip, C. Frueh, S. Lafon, R. Lyon, A. Ogale, L. Vincent, J. Weaver, Google street view: Capturing the world at street level. Computer. **43**(6), 32–38 (2010)
75. P. Zhang, M. Zhang, J. Liu, Real-time HD map change detection for crowdsourcing update based on mid-to-high-end sensors. Sensors. **21**(7), 2477 (2021)
76. Y. Zhang, X. Yuan, Y. Fang, S. Chen, UAV low altitude photogrammetry for power line inspection. ISPRS Int. J. GEO-Inf. **6**(1), 14 (2017)
77. E. Remzi, E. Alkan, A. Aydin, A comparative analysis of UAV-RTK and UAV-PPK methods in mapping different surface types. Eur. J. For. Eng. **7**(1), 12–25 (2020)
78. L. Barazzetti, F. Roncoroni, R. Brumana, M. Previtali, Georeferencing accuracy analysis of a single worldview-3 image collected over milan. XXIII ISPRS Congress. **38**, 429–434 (2016)
79. P. Daruthep, N. Sutthisangiam, in *2020 22nd International Conference on Advanced Communication Technology (ICACT)*. Development of automated processing for high-definition mapping system (IEEE, 2020), pp. 507–510
80. A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, V. Koltun, in *Proceedings of the 1st Annual Conference on Robot Learning*. CARLA: An open urban driving simulator, (2017), pp. 1–16

81. G. Rong, B. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Mozeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, S. Kim, in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. LGSVL Simulator: A high fidelity simulator for autonomous driving, (2020), pp. 1–6

Publisher's Note