Expert Systems With Applications 165 (2021) 113816

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Review

Self-driving cars: A survey

Claudine Badue [a,*], Rânik Guidolini [a], Raphael Vivacqua Carneiro [a], Pedro Azevedo [a], Vinicius B. Cardoso [a], Avelino Forechi [b], Luan Jesus [a], Rodrigo Berriel [a], Thiago M. Paixão [c], Filipe Mutz [c], Lucas de Paula Veronese [a], Thiago Oliveira-Santos [a], Alberto F. De Souza [a]

[a] Departamento de Informática, Universidade Federal do Espírito Santo, Av. Fernando Ferrari 514, 29075-910, Goiabeiras, Vitória, Espírito Santo, Brazil
[b] Coordenadoria de Engenharia Mecânica, Instituto Federal do Espírito Santo, Av. Morobá 248, 29192–733, Morobá, Aracruz, Espírito Santo, Brazil
[c] Coordenadoria de Informática, Instituto Federal do Espírito Santo, ES-010 Km-6.5, 29173-087, Manguinhos, Serra, Espírito Santo, Brazil

ARTICLE INFO

Keywords:
Self-driving cars
Robot localization
Occupancy grid mapping
Road mapping
Moving objects detection
Moving objects tracking
Traffic signalization detection
Traffic signalization recognition
Route planning
Behavior selection
Motion planning
Obstacle avoidance
Robot control

ABSTRACT

We survey research on self-driving cars published in the literature focusing on autonomous cars developed since the DARPA challenges, which are equipped with an autonomy system that can be categorized as SAE level 3 or higher. The architecture of the autonomy system of self-driving cars is typically organized into the perception system and the decision-making system. The perception system is generally divided into many subsystems responsible for tasks such as self-driving-car localization, static obstacles mapping, moving obstacles detection and tracking, road mapping, traffic signalization detection and recognition, among others. The decision-making system is commonly partitioned as well into many subsystems responsible for tasks such as route planning, path planning, behavior selection, motion planning, and control. In this survey, we present the typical architecture of the autonomy system of self-driving cars. We also review research on relevant methods for perception and decision making. Furthermore, we present a detailed description of the architecture of the autonomy system of the self-driving car developed at the Universidade Federal do Espírito Santo (UFES), named Intelligent Autonomous Robotics Automobile (IARA). Finally, we list prominent self-driving car research platforms developed by academia and technology companies, and reported in the media.

1. Introduction

Self-driving cars (also known as autonomous cars and driverless cars) have been studied and developed by many universities, research centers, car companies, and companies of other industries around the world since the middle 1980s. Important examples of self-driving car research platforms in the last two decades are the Navlab's mobile platform (Thorpe et al., 1991), University of Pavia's and University of Parma's car, ARGO (Broggi et al., 1999), and UBM's vehicles, VaMoRs and VaMP (Gregor et al., 2002).

In order to spur technology for the development of self-driving cars, the Defense Advanced Research Projects Agency (DARPA) organized three competitions in the last decade. The first, named DARPA Grand Challenge, was realized at the Mojave Desert, USA, in 2004, and required self-driving cars to navigate a 142 miles long course throughout desert trails within a 10 h time limit. All competing cars failed within the first few miles.

The DARPA Grand Challenge (Buehler et al., 2007) was repeated in 2005 and required self-driving cars to navigate a 132 miles long route through flats, dry lake beds, and mountain passes, including three narrow tunnels and more than 100 sharp left and right turns. This competition had 23 finalists and 4 cars completed the route within the allotted time limit. The Stanford University's car, Stanley (Thrun et al., 2006), claimed first place, and the Carnegie Mellon University's cars, Sandstorm and H1ghlander, finished in second and third places, respectively.

The third competition, known as the DARPA Urban Challenge (Buehler et al., 2009), was held at the former George Air Force Base, California, USA, in 2007, and required self-driving cars to navigate a 60 miles long route throughout a simulated urban environment, together with other self-driving and human driven cars, within a 6 h time limit. The cars had to obey California traffic rules. This competition

* Correspondence to: Departamento de Informática, Universidade Federal do Espírito Santo - Campus Vitória, Av. Fernando Ferrari 514, 29075-910, Goiabeiras, Vitória, Espírito Santo, Brazil.
E-mail addresses: claudine@lcad.inf.ufes.br (C. Badue), ranik@lcad.inf.ufes.br (R. Guidolini), carneiro.raphael@lcad.inf.ufes.br (R.V. Carneiro), pedro@lcad.inf.ufes.br (P. Azevedo), vinicius@lcad.inf.ufes.br (V.B. Cardoso), avelino.forechi@ifes.edu.br (A. Forechi), luan@lcad.inf.ufes.br (L. Jesus), berriel@lcad.inf.ufes.br (R. Berriel), paixao@gmail.com (T.M. Paixão), filipe.mutz@ifes.edu.br (F. Mutz), lucas.veronese@lcad.inf.ufes.br (L. de Paula Veronese), todsantos@lcad.inf.ufes.br (T. Oliveira-Santos), alberto@lcad.inf.ufes.br (A.F. De Souza).

https://doi.org/10.1016/j.eswa.2020.113816
Received 24 September 2019; Received in revised form 29 July 2020; Accepted 30 July 2020
Available online 4 August 2020

had 11 finalists and 6 cars completed the route within the allotted time limit. The Carnegie Mellon University's car, Boss (Urmson et al., 2008), claimed first place, the Stanford University's car, Junior (Montemerlo et al., 2008), finished in second, and the Virginia Tech's car, Odin (Bacha et al., 2008), came in third place. Even though these competitions presented challenges much simpler than those typically seen in everyday traffic, they have being hailed as milestones for the development of self-driving cars.

Since the DARPA challenges, many self-driving car competitions and trials have been performed. Relevant examples include: the European Land −Robot Trial (ELROB) (Schneider & Wildermuth, 2011), which has being held from 2006 to the current year; the Intelligent Vehicle Future Challenge (Xin et al., 2014), from 2009 to 2013; the Autonomous Vehicle Competition, from 2009 to 2017 (SparkFun, 2018); the Hyundai Autonomous Challenge, in 2010 (Cerri et al., 2011); the VisLab Intercontinental Autonomous Challenge, in 2010 (Broggi et al., 2012); the Grand Cooperative Driving Challenge (GCDC) (Englund et al., 2016), in 2011 and 2016; and the Proud-Public Road Urban Driverless Car Test, in 2013 (Broggi et al., 2015). At the same time, research on self-driving cars has accelerated in both academia and industry around the world. Notable examples of universities conducting research on self-driving cars comprise Stanford University, Carnegie Mellon University, MIT, Virginia Tech, FZI Research Center for Information Technology, and University of Ulm. Notable examples of companies include Google, Uber, Baidu, Lyft, Aptiv, Tesla, Nvidia, Aurora, Zenuity, Daimler and Bosch, Argo AI, Renesas Autonomy, Almotive, AutoX, Mobileye, Ambarella, Pony.ai, Idriverplus, Toyota, Ford, Volvo, and Mercedes Benz.

Although most of the university research on self-driving cars has been conducted in the United States of America, Europe and Asia, some relevant investigations have been carried out in China, Brazil and other countries. Relevant examples of self-driving car research platforms in Brazil are the Universidade Federal de Minas Gerais (UFMG)'s car, CADU (De Lima & Pereira, 2010, 2013; Dias et al., 2014; Sabbagh et al., 2010), Universidade de São Paulo's car, CARINA (Fernandes et al., 2014; Hata et al., 2017; Massera Filho et al., 2014; Shinzato et al., 2016), and the Universidade Federal do Espírito Santo (UFES)'s car, IARA (Cardoso et al., 2017; Guidolini et al., 2016, 2017; Mutz et al., 2016). IARA was the first Brazilian self-driving car to travel autonomously tens of kilometers on urban roads and highways.

To gauge the level of autonomy of self-driving cars, the Society of Automotive Engineers (SAE) International published a classification system based on the amount of human driver intervention and attentiveness required by them, in which the level of autonomy of a self-driving car may range from level 0 (the car's autonomy system issues warnings and may momentarily intervene but has no sustained car control) to level 5 (no human intervention is required in any circumstance) (SAE, 2018). In this paper, we survey research on self-driving cars published in the literature focusing on self-driving cars developed since the DARPA challenges, which are equipped with an autonomy system that can be categorized as SAE level 3 or higher (SAE, 2018).

The architecture of the autonomy system of self-driving cars is typically organized into two main parts: the perception system, and the decision-making system (Paden et al., 2016). The perception system is generally divided into many subsystems responsible for tasks such as autonomous car localization, static obstacles mapping, road mapping, moving obstacles detection and tracking, traffic signalization detection and recognition, among others. The decision-making system is commonly partitioned as well into many subsystems responsible for tasks such as route planning, path planning, behavior selection, motion planning, obstacle avoidance and control, though this partitioning is somewhat blurred and there are several different variations in the literature (Paden et al., 2016).

In this survey, we present the typical architecture of the autonomy system of self-driving cars. We also review research on relevant methods for perception and decision making.

The remainder of this paper is structured as follows. In Section 2, we present an overview of the typical architecture of the autonomy system of self-driving cars, commenting on the responsibilities of the perception system, decision making system, and their subsystems. In Section 3, we present research on important methods for the perception system, including autonomous car localization, static obstacles mapping, road mapping, moving obstacles detection and tracking, traffic signalization detection and recognition. In Section 4, we present research on relevant techniques for the decision-making system, comprising the route planning, path planning, behavior selection, motion planning, obstacle avoidance and control. In Section 5, we present a detailed description of the architecture of the autonomy system of the UFES's car, IARA. Finally, in Section 6, we list prominent self-driving car research platforms developed by academia and technology companies, and reported in the media.

2. Typical architecture of self-driving cars

In this section, we present an overview of the typical architecture of the automation system of self-driving cars and comment on the responsibilities of the perception system, decision making system, and their subsystems.

Fig. 1 shows a block diagram of the typical architecture of the automation system of self-driving cars, where the Perception and Decision Making systems (Paden et al., 2016) are shown as a collection of subsystems of different colors. The **Perception** system is responsible for estimating the State of the car and for creating an internal (to the self-driving system) representation of the environment, using data captured by on-board sensors, such as Light Detection and Ranging (LIDAR), Radio Detection and Ranging (RADAR), camera, Global Positioning System (GPS), Inertial Measurement Unit (IMU), odometer, etc., and prior information about the sensors' models, road network, traffic rules, car dynamics, etc. The **Decision Making** system is responsible for navigating the car from its initial position to the final goal defined by the user, considering the current car's State and the internal representation of the environment, as well as traffic rules and passengers' safety and comfort.

In order to navigate the car throughout the environment, the Decision Making system needs to know where the self-driving car is in it. The **Localizer** subsystem (Fig. 1) is responsible for estimating the car's State (pose, linear velocities, angular velocities, etc.) in relation to static maps of the environment (see Section 3.1). These static maps, or **Offline Maps** (Fig. 1), are computed automatically before the autonomous operation, typically using the sensors of the self-driving car itself, although manual annotations (i.e., the position of pedestrian crosswalks or of traffic lights) or editions (for removing non-static objects captured by the sensors) are usually required. A self-driving car may use for localization one or more Offline Maps, such as occupancy grid maps, remission maps or landmark maps. We survey the literature on methods for generating Offline Maps in Section 3.2.

Information regarding the rules and regulations on how the self-driving car may move in the roads and highways (direction of traffic, maximum speed, lane demarcation, etc.) are also essential to the Decision Making system. This information is typically embedded in road maps, which represent it in such maps using geometrical and topological properties. We survey the literature on road mapping in Section 3.3.

The Localizer subsystem receives as input the Offline Maps, Sensors' data and the Odometry of the self-driving car, and computes as output the car's State (Fig. 1). It is important to note that, although GPS may help the localization process, it alone is not enough for proper localization in urban environments due to interferences caused by tall trees, buildings, tunnels, etc., that makes GPS positioning unreliable. We survey the literature on localization techniques in Section 3.1.

The **Mapper** subsystem receives as input the Offline Maps and the self-driving car's State, and generates as output the online map. This

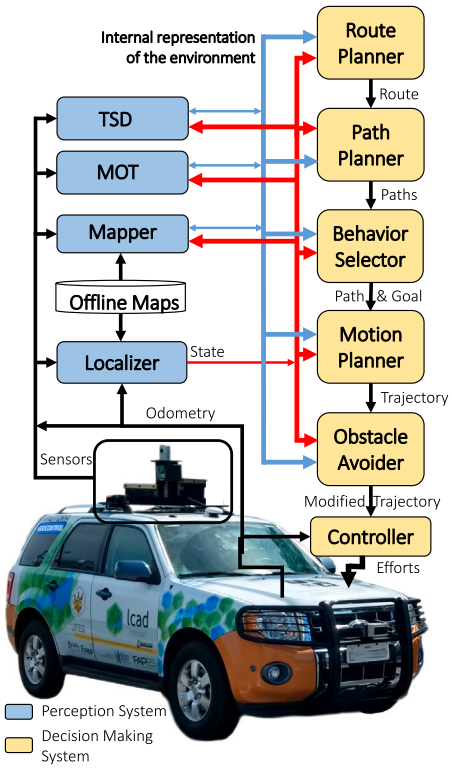


Fig. 1. Overview of the typical architecture of the automation system of self-driving cars. TSD denotes Traffic Signalization Detection and MOT denotes Moving Objects Tracking.

online map is typically a merge of information present in the Offline Maps and an occupancy grid map computed online using sensors' data and the current car's State. We survey the literature on methods for computing the online map in Section 3.2. It is desirable that the online map contains only a static representation of the environment, since this may help the operation of some subsystems of the Decision Making system.

Information regarding the pose and velocity of moving obstacles is also essential to the Decision Making system. This information enables this system to make decisions that avoid collision with moving obstacles. It also allows the removal of moving obstacles from the online map. The **Moving Objects Tracker** subsystem, or **MOT** (Fig. 1), receives the Offline Maps and the self-driving car's State, and detects and tracks, i.e., calculates the pose and velocity of, the nearest moving obstacles (e.g., other vehicles and pedestrians). We survey the literature on methods for moving objects detection and tracking in the context of self-driving cars in Section 3.4.

Horizontal (i.e. lane markings) and vertical (i.e. speed limits, traffic lights, etc.) traffic signalization must be recognized and obeyed by self-driving cars. The **Traffic Signalization Detector** subsystem, or **TSD** (Fig. 1), is responsible for the detection and recognition of traffic signalization. It receives the Sensors' data and the car's State, and detects the position of traffic signalizations and recognizes their class or status. We survey the literature on methods for traffic signalization detection and recognition in Section 3.5.

Given a Final Goal defined in the Offline Maps by the user, the **Route Planner** subsystem computes a Route, W, in the Offline Maps, from the current self-driving car's State to the Final Goal. A Route is a sequence of way points, i.e. $W = \{w_1, w_2, \ldots, w_{|W|}\}$, where each way point, w_i, is a coordinate pair, i.e. $w_i = (x_i, y_i)$, in the Offline Maps. We survey the literature on methods for route planning in Section 4.1.

Given a Route, the **Path Planner** subsystem computes, considering the current self-driving car's State and the internal representation of the environment as well as traffic rules, a set of Paths, $P = \{P_1, P_2, \ldots, P_{|P|}\}$. A Path is a sequence of poses, i.e. $P_j = \{p_1, p_2, \ldots, p_{|P|}\}$, where each pose, p_i, is a coordinate pair in the Offline Maps and the desired car's orientation at the position defined by this coordinate pair, i.e. $p_i = (x_i, y_i, \theta_i)$. We survey the literature on methods for path planning in Section 4.2.

The **Behavior Selector** subsystem is responsible for choosing the current driving behavior, such as lane keeping, intersection handling, traffic light handling, etc. It does so by selecting a Path, P_j, in P, a pose, p_g, in P_j a few seconds ahead of the current self-driving car's State (about 5 s), which is the decision horizon, and the desired velocity at this pose. The pair pose in P_j and associated velocity is called $Goal_g = (p_g, v_g)$. The Behavior Selector chooses a Goal considering the current driving behavior and avoiding collisions with static and moving obstacles in the environment within the decision horizon time frame. We survey the literature on methods for behavior selection in Section 4.3.

The **Motion Planner** subsystem is responsible for computing a Trajectory, T, from the current self-driving car's State to the current Goal, which follows the Path defined by the Behavior Selector, satisfies car's kinematic and dynamic constraints, and provides comfort to the passengers. A Trajectory $T = \{c_1, c_2, \ldots, c_{|T|}\}$ may be defined as a sequence of commands, $c_k = (v_k, \varphi_k, \Delta t_k)$, where v_k is the desired velocity at time k, φ_k is the desired steering angle at time k, and Δt_k is the duration of c_k; there are other forms of defining trajectories, however (see Section 4.4). A Trajectory takes the car from its current State to the current Goal smoothly and safely. We survey the literature on methods for motion planning in Section 4.4.

The **Obstacle Avoider** subsystem receives the Trajectory computed by the Motion Planner and changes it (typically reducing the velocity), if necessary, to avoid collisions. There is no much literature on methods for performing the functions of the Obstacle Avoider subsystem. We discuss some relevant literature on this subject in Section 4.5.

Finally, the **Controller** subsystem receives the Motion Planner trajectory, eventually modified by the Obstacle Avoider subsystem, and computes and sends Effort commands to the actuators of the steering wheel, throttle and brakes in order to make the car execute the Modified Trajectory as best as the physical world allows. We survey the literature on methods for low level car control in Section 4.6.

In the following, we detail each one of these subsystems and the techniques used to implement them and their variants, grouped within perception and decision-making systems.

3. Self-driving cars' perception

In this section, we present research on important methods proposed in the literature for the perception system of self-driving cars, including methods for localization, offline obstacle mapping, road mapping, moving obstacle tracking, and traffic signalization detection and recognition.

3.1. Localization

The **Localizer** subsystem is responsible for estimating the self-driving car pose (position and orientation) relative to a map or road (e.g., represented by curbs or road marks). Most general-purpose localization subsystems are based on GPS. However, by and large, they are not applicable to urban self-driving cars, because the GPS signal cannot be guaranteed in occluded areas, such as under trees, in urban canyons (roads surrounded by large buildings) or in tunnels.

Various localization methods that do not depend on GPS have been proposed in the literature. They can be mainly categorized into three classes: LIDAR-based, LIDAR plus camera-based, and camera-based. LIDAR-based localization methods rely solely on LIDAR sensors, which offer measurement accuracy and easiness of processing. However, despite the LIDAR industry efforts to reduce production costs, they still

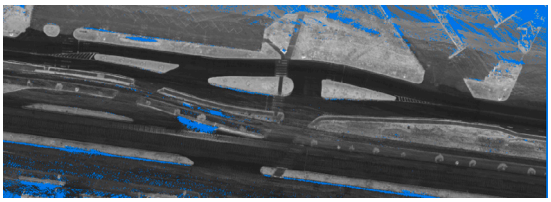
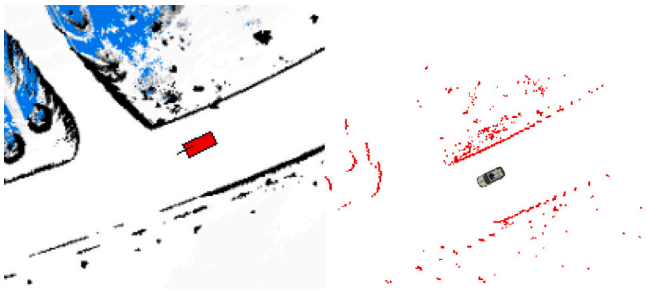


Fig. 2. Remission map.



(a) (b)

Fig. 3. Localization method proposed by Veronese et al. (2016). (a) Offline occupancy grid-map — the red rectangle is the car's localization, black cells contain obstacles, white cells are obstacle-free, and blue cells are regions untouched by sensors during mapping. (b) Online occupancy grid-map. The online map is matched against the offline map to compute the self-driving car's precise localization.

have a high price if compared with cameras. In typical LIDAR plus camera-based localization methods, LIDAR data is used only to build the map, and camera data is employed to estimate the self-driving car's position relative to the map, which reduces costs. Camera-based localization approaches are cheap and convenient, even though typically less precise and/or reliable.

3.1.1. LIDAR-based localization

Levinson and Thrun (2010) proposed a localization method that uses offline grid maps of reflectance intensity distribution of the environment measured by a LIDAR scanner (remission grid maps, Fig. 2); they have used the Velodyne HDL-64E LIDAR in their work. An unsupervised calibration method is used to calibrate the Velodyne HDL-64E' laser beams so that they all respond similarly to the objects with the same brightness, as seen by the LIDAR. A 2-dimension histogram filter (Thrun et al., 2005) is employed to estimate the autonomous vehicle position. As usual, the filter is comprised of two parts: the motion update (or prediction), to estimate the car position based on its motion, and the measurement update (or correction), to increase confidence in our estimate based on sensor data. In the motion update, the car motion is modeled as a random walk with Gaussian noise drift from a dead reckoning coordinate system (computed using the inertial update of an Applanix POS LV-420 position and orientation system) to the global coordinate system of the offline map. In the measurement step, they use, for different displacements, the similarity between the remission map computed online, with the remission map computed offline. Each displacement corresponds to one cell of the histogram of the histogram filter. To summarize the histogram into a single pose estimate, they use the center of mass of the probability distribution modeled by the histogram. The authors do not describe how they estimate the orientation, though. Their method has shown a Root Mean Squared (RMS) lateral error of 9 cm and a RMS longitudinal error of 12 cm.

Veronese et al. (2015) proposed a MCL localization method that compares satellite aerial maps with remission maps. Aerial maps are downloaded offline from sources in the Internet, like OpenStreetMap, and remission maps are built online from LIDAR reflectance intensity data. The MCL algorithm is employed to estimate car's pose by matching remission maps to aerial maps using the Normalized Mutual Information (NMI) measure to calculate the particles likelihood. The method was evaluated on a 6.5 km dataset collected by the self-driving car IARA and achieved position estimation accuracy of 0.89 m. One advantage of this method is that it does not require building a map specifically for the method.

Hata and Wolf (2015) proposed a localization method based on road feature detection. Their curb detection algorithm uses ring compression analysis and least trimmed squares is used to analyze the distance between consecutive concentric measurements (or rings) formed by a multilayer LIDAR (Velodyne HDL-32E) scan. The road marking detection algorithm uses Otsu thresholding (Otsu, 1979) to analyze LIDAR reflectance intensity data. Curb and road marking features are stored in a grid map. A Monte Carlo Localization (MCL) algorithm is employed to estimate the car pose by matching road features extracted from multilayer LIDAR measurements to the grid map. The method was evaluated on the self-driving car, CARINA (Fernandes et al., 2014), and has shown lateral and longitudinal localization estimation errors of less than 0.30 m.

Rohde et al. (2016) proposed a multilayer adaptive Monte Carlo Localization (ML-AMCL) method that operates in combination with 3D point registration algorithms. For estimating the car pose, horizontal layers are extracted from 3D LIDAR measurements and separate AMCL instances are used to align layers with a 2D projection of a 3D point cloud map built using 3D point registration algorithms. For every pose estimate, a consistency check against a series of odometry measurements is performed. Consistent pose estimates are fused to a final pose estimate. The method was evaluated on real world data and achieved position estimation errors of 0.25 m relative to the GPS reference. Their map is, however, expensive to store, since it is a 3D map.

Veronese et al. (2016) proposed a localization method based on the MCL algorithm that corrects particles' poses by map-matching between 2D online occupancy grid-maps and 2D offline occupancy grid-maps, as illustrated in Fig. 3. Two map-matching distance functions were evaluated: an improved version of the traditional Likelihood Field distance between two grid-maps, and an adapted standard Cosine distance between two high-dimensional vectors. An experimental evaluation on the self-driving car IARA demonstrated that the localization method is able to operate at about 100 Hz using the Cosine distance function, with lateral and longitudinal errors of 0.13 m and 0.26 m, respectively.

Wolcott and Eustice (2017) proposed a probabilistic localization method that models the world as a multiresolution map of mixture of Gaussians. Their Gaussian mixture maps represent the height and reflectance intensity distribution of the environment measured by LIDAR scanners (Velodyne HDL-32E). An Extended Kalman filter (EKF) localization algorithm is used to estimate the car's pose by registering 3D point clouds against the Gaussian mixture multiresolution-maps. The method was evaluated on two self-driving cars in adverse weather conditions and presented localization estimation errors of about 0.15 m.

3.1.2. LIDAR plus camera-based localization

Some methods use LIDAR data to build a map, and camera data to estimate the localization of the self-driving car relative to this map. Xu et al. (2017) proposed a localization method that matches stereo images to a 3D point-cloud map. The map was generated by a mapping company and it is composed of geometric data (latitude, longitude and altitude) and LIDAR reflectance intensity data acquired from odometer, RTK-GPS, and 2D LIDAR scanners. Xu et al. transform the 3D-points of the map from the real-world coordinate system to the camera coordinate system, and extract depth and intensity images from them. A MCL algorithm is used to estimate the car localization by matching stereo depth and intensity images taken from the car's

camera to depth and intensity images extracted from the 3D point-cloud map. The method was evaluated on real world data and presented localization estimation errors between 0.08 m and 0.25 m.

Viswanathan et al. (2016) proposed a method for autonomous vehicle localization that matches ground panoramic-images to satellite images captured in different seasons of the year. In their method, LIDAR data is classified into ground/non-ground categories. Next, ground images captured by a panoramic camera in the autonomous vehicle are segmented into ground/non-ground regions using the LIDAR data, and then warped to obtain a bird's-eye view. The satellite image is also segmented into ground/non-ground regions using k-means clustering. A MCL is then used to estimate the pose by matching bird's-eye images to the satellite images. The method was evaluated on the NavLab11 autonomous vehicle and achieved position estimation errors between 3 m and 4.8 m.

3.1.3. Camera-based localization

Some methods rely mainly on camera data to localize self-driving cars. Brubaker et al. (2015) proposed a localization method based on visual odometry and road maps. They use the OpenStreetMap, extracting from it all crossings and all drivable roads (represented as piece-wise linear segments) connecting them in the area of interest. They, then, build a graph-based representation of this road map and a probabilistic model of how the car traverses this graph. Using this probabilistic model and visual odometry measurements, they estimate the car displacement relative to the road map.

A recursive Bayesian filtering algorithm is used to perform inferences in the graph by exploiting its structure and the model of how the car moves, as measured by the visual odometry. This algorithm is able to pinpoint the car's position in the graph by increasing the probability that the current pose lies in a point of the graph that is correlated with latest car movements (distance traveling straight and recent curves) and by decreasing the probability that it is in a point that is not correlated. The localization method was evaluated on the KITTI visual odometry dataset and was able to localize the autonomous vehicle, after 52 s of driving, with an accuracy of 4 m on an 18 km^2 map containing 2150 km of drivable roads.

Some methods use camera data to build a feature map. Ziegler, Lategahn et al. (2014) describe the localization methods used by the self-driving car Bertha to drive autonomously on the Bertha-Benz-Memorial-Route. Two complementary vision based localization techniques were developed, named Point Feature based Localization (PFL) and Lane Feature based Localization (LFL). In PFL, the current camera image is compared with images of a sequence of camera images that is acquired previously during mapping using DIRD descriptors extracted from them. A global location estimate is recovered from the global position of the images captured during mapping. In LFL, the map, computed semi-automatically, provides a global geometric representation of road marking features (horizontal road signalization). The current camera image is matched against the map by detecting and associating road marking features extracted from a bird's-eye view of the camera image with horizontal road signalization stored in the map. Location estimates obtained by PFL and LFL are, then, combined by a Kalman filter (the authors do not provide an estimate of the combined localization error). Localization methods similar to LFL were proposed by Jo et al. (2015), Suhr et al. (2016), and Vivacqua et al. (2017).

Some methods employ camera data to construct a feature map, but adopt alternative types of features. Radwan et al. (2016) proposed a localization method based on textual feature detection. Off-the-shelf text extraction techniques are used to identify text labels in the environment. A MCL algorithm is employed to integrate multiple observations. The method was evaluated on real world data and presented location estimation errors between 1 m and 25 m. Spangenberg et al. (2016) proposed the use of pole-like landmarks as primary features, because they are distinctive, long-term stable, and detectable by stereo cameras. Furthermore, they allow a memory efficient map representation. The feature detection is performed mainly by a stereo camera. Localization is performed by a MCL algorithm coupled with a Kalman filter for robustness and sensor fusion. The method was evaluated on an autonomous vehicle and achieved position estimation errors between 0.14 m and 0.19 m.

Some methods employ neural networks to localize self-driving cars (Lyrio et al., 2015; Oliveira et al., 2017). They consist of correlating camera images and associated global positions. In the mapping phase, the neural network builds a representation of the environment. For that, it learns a sequence of images and global positions where images were captured, which are stored in a neural map. In the localization phase, the neural network uses previously acquired knowledge, provided by the neural map, to estimate global positions from currently observed images. These methods present error of about some meters and have difficulty in localizing self-driving cars on large areas.

3.2. Offline and online mapping of unstructured environments

The offline and online **Mapper** subsystems are responsible for computing maps of the environment where the self-driving car operates. These subsystems are fundamental for allowing them to navigate on unstructured environments without colliding with static obstacles (e.g., signposts, curbs, etc.).

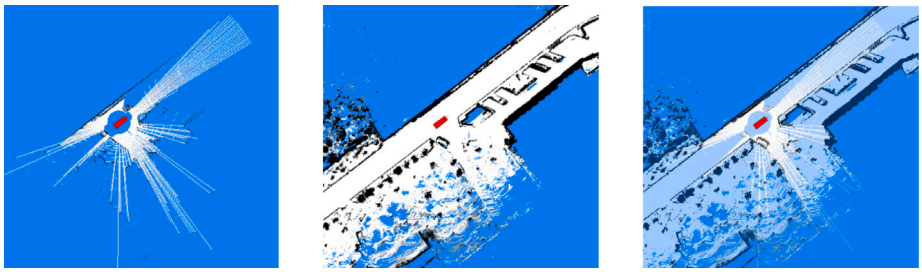
Representations of the environment are often distinguished between topological (Cummins & Newman, 2008; Forechi et al., 2018; Milford & Wyeth, 2012) and metric (Hornung et al., 2013; Mutz et al., 2016; Schaefer et al., 2018). Topological representations model the environment as a graph, in which nodes indicate significant places (or features) and edges denote topological relationships between them (e.g., position, orientation, proximity, and connectivity). The resolution of these decompositions depends on the structure of the environment.

Metric representations usually decompose the environment into regularly spaced cells. This decomposition does not depend on the location and shape of features. Spatial resolution of metric representations tends to be higher than that of topological representations. This makes them the most common space representation. For a review on the main vision-based methods for creating topological representations, readers are referred to Garcia-Fidalgo and Ortiz (2015). Here, we discuss the most important methods for computing metric representations, which can be further subdivided into grid representations with regular spacing resolution and varied spacing resolution.

3.2.1. Regular spacing metric representations

For self-driving cars, one of the most common representations of the environment is the Occupancy Grid Map (OGM), proposed by Moravec and Elfes (1985). An OGM discretizes the space into fixed size cells, usually of the order of centimeters. Each cell contains the probability of occupation of the region associated with it. The occupancy probability is updated independently for each cell using sensor data. 3D sensor measurements that characterize the environment can be projected onto the 2D ground plane for simplicity and efficiency purposes. The assumption of independence of the occupancy probability of each cell makes the OGM algorithm implementation fast and easy (Thrun et al., 2005). However, it generates a sparse space representation, because only those cells touched by the sensor are updated (Kim & Kim, 2013). Fig. 4 shows examples of OGMs computed by the self-driving car IARA. In the maps of this figure, shades of gray represents the occupancy probability of cells, with black being the maximum occupancy probability and white being the minimum occupancy probability, and blue represents cells that were not observed by sensors yet.

Thrun and Montemerlo (2006) presented the GraphSLAM algorithm. GraphSLAM is an offline Simultaneous Localization And Mapping (SLAM) algorithm, which extracts from sensor data a set of soft constraints, represented by a sparse graph. It is able to obtain the map of the environment and the robot path that was followed during sensor data capture by resolving these constraints into a globally consistent



(a) (b) (c)

Fig. 4. Examples of OGMs computed by the self-driving car IARA. Shades of gray represent the occupancy probability of map cells and blue represents cells that were not observed by sensors yet. Black represents maximal occupancy probability, while white represents minimal occupancy probability. (a) The instantaneous OGM that is computed online using a single LIDAR point cloud. (b) The offline OGM that is constructed offline using data obtained by various sensors and stores the most consolidated information that IARA has about the world. (c) The online map that is used by the motion planner subsystem. It is updated online by merging the previous ones, where the dark blue represents the cells in common in both maps and the light blue represents the cells that exist in the first but not in the second one.

estimate of robot poses. The constraints are related to the properties of the data captured by the sensors (e.g. distance and elapsed time between two consecutive odometry measurements) and are generally nonlinear but, in the process of resolving them, they are linearized and the resulting least squares problem is solved using standard optimization techniques. Once the robot poses are estimated, the map can be built from sensor data using probabilistic algorithms for OGM computation.

Mutz et al. (2016) employed GraphSLAM and data captured by an odometer, a 3D LIDAR Velodyne HDL-32E, IMU, and a low-cost GPS to build OGMs and for self-driving cars. In their work, GPS data is used to identify and remove systematic errors in odometer data. GPS data is also employed to detect revisited regions (loop closure), and a Generalized Iterative Closest Point (GICP) algorithm (Segal et al., 2009) is employed to estimate displacements between poses in subsequent visits. Data from GPS, IMU, calibrated odometer, and loop closures are used to introduce restrictions to the GraphSLAM algorithm, which calculates the most likely set of poses given sensor data. Finally, 3D LIDAR data and their poses, calculated by GraphSLAM, are used (offline) to build an OGM. Besides the GraphSLAM bundle adjustment optimization, there are other algorithms for offline mapping, such as gradient descent, conjugate gradient and Levenberg Marquardt (Thrun et al., 2005).

During operation, a self-driving car requires an offline map for localization. It may also require an online map for applications where the environment contains moving obstacles. These two maps can be merged for improved operation safety (Teixeira et al., 2018).

SLAM can be computed online using FastSLAM (Stentz et al., 2003). FastSLAM was originally designed for landmark maps but further extended for OGMs. The original FastSLAM uses a particle filter algorithm to estimate the self-driving car's path and an EKF algorithm for the landmarks' positions, while the extended FastSLAM employs a particle filter to estimate both the car's path and the occupancy probability of OGM's cells. There are other algorithms for online grid mapping (Hess et al., 2016; Kohlbrecher et al., 2011).

3.2.2. *Varied spacing metric representations*

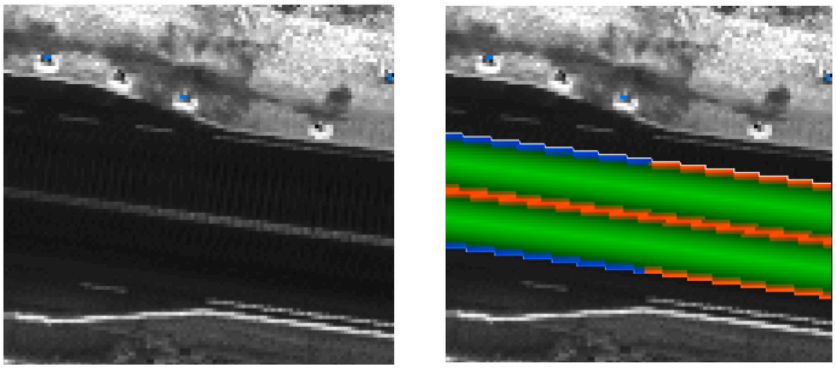
An alternative metric representation of the environment is the Octree map, proposed by Hornung et al. (2013), which stores information with varied 3D resolutions. Compared to OGMs with varied 3D resolutions, OctoMaps (Octree-based Maps) store only the space that is observed and, consequently, are more memory efficient. They are, however, computationally intensive (Chen & Shen, 2017), which makes them unsuitable for self-driving cars, considering the hardware currently available. In addition, the typical self-driving car can be modeled as a parallelepiped, or a series of interconnected parallelepipeds, and the map may only contain information pertained to objects of the environment that are obstacles for the movements of these parallelepipeds.

Another alternative metric representation is a hybrid map proposed by Droeschel et al. (2017), which stores occupancy and distance measurements with varied resolutions. For this, measurements are stored in grid cells of increasing size from the center of the car. Thus, computational efficiency is gained by having a high resolution in the proximity of the sensor and a lower resolution as the distance from the sensor grows. This follows characteristics of some sensors with respect to distance and measurement density (e.g., angular resolution of laser sensors).

Traditional methods for computing OGMs are constrained by the assumption that the probability of occupancy of each cell of the map is modeled as an independent random variable (Thrun et al., 2005). Intuitively, this assumption is not true, because real environments have some inherent structure. An alternative metric representation is the Gaussian Process Occupancy Map (GPOM) proposed by Doherty et al. (2016). A GPOM uses a Gaussian Process (GP) to learn the structure of the environment from sparse sensor measurements in a training dataset and, subsequently, estimate the probability of occupancy of cells that were not directly intercepted by the sensor. Experiments have shown that localization errors are more than three times lower as those estimated by a particle filter localization and OGM (Hata et al., 2016). However, this inference carries a high computational cost of $O\left(N^3\right)$, where N is the number of samples in the training dataset, and, therefore, it is not directly applicable to large-scale scenarios of self-driving cars (Kim & Kim, 2013).

Another alternative metric representation is the Hilbert map proposed by Ramos and Ott (2016). Hilbert maps represent the probability of occupancy with a linear discriminative model that operates on a high-dimensional feature vector and projects observations into a Hilbert space. The model can be trained and updated online using Stochastic Gradient Descent (SGD). Experiments showed that a Hilbert map presents accuracy comparable to a Gaussian Process Occupancy Map (GPOM), but it has time complexity linear with the number of samples in the training dataset.

A further alternative metric representation is the Discrete Cosine Transform (DCT) map proposed by Schaefer et al. (2018). A DCT map assigns to each point of the space a LIDAR decay rate, which models the local permeability of the space for laser rays. In this way, the map can describe obstacles of different laser permeability, from completely opaque to completely transparent. A DCT map is represented in the discrete frequency domain and converted to a continuously differentiable field in the position domain using the continuous extension of the inverse DCT. DCT maps represent LIDAR data more accurately than OGM, GPOM, and Hilbert map regarding the same memory requirements. Nonetheless, the above continuous metric representations are still slower than OGM and not widely applicable to large-scale and real-time self-driving scenarios yet.



(a) (b)

Fig. 5. Metric road map. (a) Crop of a remission grid map. (b) Same as (a) with superimposed road grid map. The value of each cell of the road grid map represents different classes of features. Red cells represent a Solid Line close to the lane boundary, blue cells represent a Broken Line close to the lane boundary, different shades of green represent different distances from the map cell to the center of the lane.

3.3. Road mapping

The mapping techniques presented above can be used to map unstructured environments and make it possible the operation of self-driving cars in any kind of flat terrain. However, for autonomous operation in roads and highways, where there are rules consubstantiated in markings in the floor and other forms of traffic signalization, self-driving cars need road maps. The **Road Mapper** subsystem is responsible for managing information about roads and lanes, and for making them available in maps with geometrical and topological properties.

Road maps can be created manually from aerial images. However, the high cost of maintaining them through manual effort makes this approach unviable for large scale use. Because of that, methods for automated generation of road maps from aerial images have been proposed.

Here, we discuss the most important road map representations and the methods used for road map creation.

3.3.1. Metric representations

A simple metric representation for a road map is a grid map, which discretizes the environment into a matrix of fixed size cells that contain information about the roads. However, a grid representation might require a wasteful use of memory space and processing time, since usually most of the environment where self-driving cars operate is not composed of roads, but buildings, free space, etc.

Carneiro et al. (2018) proposed a metric road map (Fig. 5), a grid map, where each 0.2 m × 0.2 m-cell contains a code that, when nonzero, indicates that the cell belongs to a lane. Codes ranging from 1 to 16 represent relative distances from a cell to the center of the lane, or the type of the different possible lane markings (broken, solid, or none) present in the road. To save memory space, these maps are stored in a compacted form, where only non-zero cells are present. The authors used Deep Neural Networks (DNNs) to infer the position and relevant properties of lanes with poor or absent lane markings. The DNN performs a segmentation of LIDAR remission grid maps into road grid maps, assigning the proper code (from 1 to 16) to each map cell. A dataset of tens of kilometers of manually marked road lanes was used to train the DNN, allowing it achieve an accuracy of 83.7%, which proved to be sufficient for real-world applications.

3.3.2. Topological representations

Sequences of waypoints, representing the center of the lanes of the roads of interest, are an alternative simple topological road map representation. They can be defined manually, or captured semi automatically or automatically from OGM, aero-photogrammetry or satellite images. For the 2005 DARPA Grand Challenge, DARPA used a road map representation named Route Data Definition File (RDDF) (DARPA, 2005), which is a formatted file that contains waypoint coordinates and other associated information (latitude, longitude, lateral boundary offset, and course speed) that specified the path for the operation of the competing self-driving cars.

A more sophisticated road map representation depicts the environment as a graph-like model, in which vertices denote places and edges denote topological relationships between them. Topological maps can hold more complex information, including multiple lanes, lane crossings, and lane mergers. For the 2007 DARPA Urban Challenge, it was proposed the Route Network Definition File (RNDF) (DARPA, 2007), which is a topological map defined as a formatted file that specifies road segments for the operation of the competing self-driving cars. In a RNDF, the road network includes one or more segments, each of which comprises one or more lanes. A segment is characterized by the number of lanes, street name, and speed limit. A lane is characterized by the width of the lane, lane markings, and a set of waypoints. Connections between lanes are characterized by exit and entry waypoints.

Urmson et al. (2008) used a graph model of the DARPA RNDF for the self-driving car Boss (Carnegie Mellon University's car that claimed first place in the 2007 DARPA Urban Challenge). Each node in the graph denotes a waypoint and directional edges denote lanes that connect the node to all other waypoints it can reach. Costs are assigned to the edges based on a combination of several factors, including expected time to traverse the lane associated to the edge, length of the lane, and complexity of the environment. The authors used manual annotation of road shapes extracted from aerial imagery in order to create a road map for the self-driving car Boss. The obtained local road shapes were accurate; however, global positions were not so accurate due to the image resolution and the global registration method.

Ramm et al. (2011) proposed the OpenStreetMap (OSM). OSM is a collaborative project to create a free editable map of the world. Its creation and growth have been motivated by restrictions on use or availability of map information across much of the world and by the advent of inexpensive portable satellite navigation devices. OSM models the environment with topological maps using three primitive elements, namely: nodes, ways and relations. Nodes denote geographical points, ways denote lists of nodes (polylines), and relations consist of any number of members that may be of any of the three types and have specified roles. Other road properties, like the driving direction and the number of lanes, are given as properties of the elements.

Bender et al. (2014) proposed a highly detailed topological road map, called lanelet map, for the self-driving car Bertha. The lanelet map includes both geometrical and topological features of road networks, such as roads, lanes, and intersections, using atomic interconnected

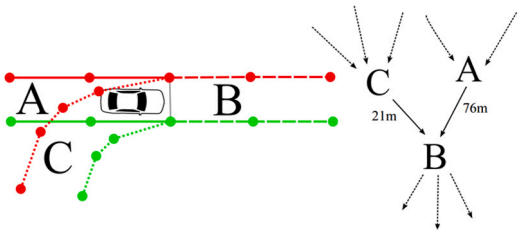


Fig. 6. A graph model of a lanelet map used by the self-driving car Bertha (Bender et al., 2014). Red and green dots denote the polylines of left and right bounds of lanelets A, B and C, respectively. The graph shows the merge of A and C into B.

drivable road segments, called lanelets, illustrated in Fig. 6. The geometry of a lanelet is defined by a left and a right bound, each one corresponding to a list of points (polyline). This representation implicitly defines the width and shape of each lane and its driving orientation. The adjacency of lanelets composes a weighted directed graph, in which each lanelet denotes a vertex and the length of a lanelet denotes the weight of its outgoing edges. Other elements describe constraints, such as speed limits and traffic regulations, like crossing and merging rights. The authors adopted manual annotation of all elements and properties of their lanelet maps for the self-driving car Bertha. Virtual top-view images were used as a foundation for the manual annotation of lanelets using the OSM format and the Java OSM editor. The lanelet map was successfully tested throughout an autonomous journey of 103 km on the historic Bertha Benz Memorial Route.

Bastani et al. (2018) proposed the Road Tracer method that seeks to produce a road map directly from the output of a CNN. It uses an iterative graph construction process that adds individual road segments, one at a time, and uses the CNN to decide on the next segment to be added. The test on aerial images covering 24 km^2 of 15 cities achieved an average error of 5% on a junction-by-junction matching metric.

High-Definition (HD) maps are a new generation of topological maps that are being used by some existing experimental self-driving cars. HD maps have high-precision (centimeter-level precision) and contain rich information, such as lane positions, road boundaries, and road curvatures. Since the cost incurred to create HD maps is significant, there are platforms available to provide HD maps as a service. Zoller (2018) assessed and ranked the top vendors, namely Google, HERE, TomTom, and Apple.

Besides the above-mentioned map generation methods, many other methods have been proposed for automated generation of road maps from aerial images. Wegner et al. (2015) used higher-order Conditional Random Fields (CRF) to model the structures of the road network by segmenting images into superpixels and adding paths to connect these superpixels. Mnih and Hinton (2010) used Convolutional Neural Networks (CNNs) to obtain road segments from satellite images. They improved the predictive performance by using unsupervised learning methods for initializing the feature detectors and taking advantage of the local spatial coherence of the output.

There are several techniques for lane detection in images captured by top-view or front-facing cameras. For reviews on these topics, readers are referred to Hillel et al. (2014) and Yenikaya et al. (2013).

Aeberhard et al. (2015) use ground grid maps, in which each cell represents the probability of a ground location with high reflectance intensity, for the BMW's self-driving car. The ground grid maps are used to extract road boundaries using a second-degree polynomial model. Lane localization is used in conjunction with a HD map in order to obtain a higher level of understanding of the environment. The map consists of two layers: a semantic geometric layer and a localization layer. The semantic geometric layer contains the lane geometry model and the high-level semantic information, such as lane connectivity. The localization layer contains lane markings and road boundaries, which, together with GPS and car odometry, can be used to match the car onto the map.

Lee et al. (2018) also used LIDAR remission data to detect lane markings and camera images, in case lane makings are not well defined. The lane markings on the road used during experiments was made to look good with headlight at night by using a special paint with good reflectance intensity. With this approach, road markings could also be detected with LIDAR, even in the case of changes in illumination due to rain or shadows. The lane marking detection technique based on camera image was run only in vulnerable situations (e.g., backlight and low light). This approach was successfully tested in a course 2 km in Seoul, Korea.

3.4. Moving Objects Tracking

The **Moving Objects Tracker (MOT)** subsystem (also known as Detector and Tracker of Moving Obstacles – DATMO) is responsible for detecting and tracking the pose of moving obstacles in the environment around the self-driving car. This subsystem is essential for enabling self-driving cars to make decisions about how to behave to avoid collision with potentially moving objects (e.g., other vehicles and pedestrians).

Moving obstacles' positions over time are usually estimated from data captured by ranging sensors, such as LIDAR and RADAR, or stereo and monocular cameras. Images from monocular cameras are useful to provide rich appearance information, which can be explored to improve moving obstacle hypotheses. To cope with uncertainty of sensor measurements, Bayes filters (e.g., Kalman and particle filter) are employed for state prediction.

Various methods for MOT have been proposed in the literature. They can be mainly categorized into six classes: traditional, model based, stereo vision based, grid map based, sensor fusion based, and deep learning based. Here, we present the most recent and relevant ones published in the last ten years. For earlier works, readers are referred to Bernini et al. (2014), Girao et al. (2016) and Petrovskaya et al. (2012).

3.4.1. Traditional based MOT

Traditional MOT methods follow three main steps: data segmentation, data association, and filtering (Petrovskaya et al., 2012). In the data segmentation step, sensor data are segmented using clustering or pattern recognition techniques. In the data association step, segments of data are associated with targets (moving obstacles) using data association techniques. In the filtering phase, for each target, a position is estimated by taking the geometric mean of the data assigned to the target. Position estimates are usually updated by Kalman or particle filters. Amaral et al. (2015) propose a traditional method for detection and tracking of moving vehicles using 3D LIDAR sensor. The 3D LIDAR point cloud is segmented into clusters of points using the Euclidean distance. Obstacles (clusters) observed in the current scan sensor are associated with obstacles observed in previous scans using a nearest neighbor algorithm. States of obstacles are estimated using a particle filter algorithm. Obstacles with velocity above a given threshold are considered moving vehicles. Zhang et al. (2013) build a cube bounding box for each cluster and use box dimensions for distinguishing whether a cluster is a vehicle or not. Data association is solved by an optimization algorithm. A Multiple Hypothesis Tracking (MHT) algorithm is employed to mitigate association errors. Hwang et al. (2016) use images captured by a monocular camera to filter out 3D LIDAR points that do not belong to moving objects (pedestrians, cyclists and vehicles). Once filtered, object tracking is performed based on a segment matching technique using features extracted from images and 3D points.

3.4.2. Model based MOT

Model-based methods directly infer from sensor data using physical models of sensors and geometric models of objects, and employing non-parametric filters (e.g., particle filters) (Petrovskaya et al., 2012). Data segmentation and association steps are not required, because geometric object models associate data to targets. Petrovskaya and Thrun (2009) present the model-based method for detection and tracking of moving vehicles adopted by the self-driving car, Junior (Montemerlo et al., 2008) (Stanford University's car that finished in second place in the 2007 DARPA Urban Challenge). Moving vehiclwe e hypotheses are detected using differences over LIDAR data between consecutive scans. Instead of separating data segmentation and association steps, new sensor data are incorporated by updating the state of each vehicle target, which comprises vehicle pose and geometry. This is achieved by a hybrid formulation that combines Kalman filter and Rao-Blackwellized Particle Filter (RBPF). The work of Petrovskaya and Thrun (2009) was revised by He et al. (2016) that propose to combine RBPF with Scaling Series Particle Filter (SSPF) for geometry fitting and for motion estimate throughout the entire tracking process. The geometry became a tracked variable, which means that its previous state is also used to predict the current state. Vu and Aycard (2009) propose a model-based MOT method that aims at finding the most likely set of tracks (trajectories) of moving obstacles, given laser measurements over a sliding window of time. A track is a sequence of object shapes (L-shape, I-shape and mass point) produced over time by an object satisfying the constraints of a measurement model and motion model from frame to frame. Due to the high computational complexity of such a scheme, they employ a Data Driven Markov chain Monte Carlo (DD-MCMC) technique that enables traversing efficiently in the solution space to find the optimal solution. DD-MCMC is designed to sample the probability distribution of a set of tracks, given the set of observations within a time interval. At each iteration, DD-MCMC samples a new state (set of tracks) from the current state following a proposal distribution. The new candidate state is accepted with a given probability. To provide initial proposals for the DD-MCMC, dynamic segments are detected from laser measurements that fall into free or unexplored regions of an occupancy grid map and moving obstacle hypotheses are generated by fitting predefined object models to dynamic segments. Wang et al. (2015) adopt a similar method to the model-based one, but they do not assume prior categories for moving objects. A Bayes filter is responsible for joint estimation of the pose of the sensor, geometry of static local background, and dynamics and geometry of objects. Geometry information includes boundary points obtained with a 2D LIDAR. Basically, the system operates by iteratively updating tracked states and associating new measurements to current targets. Hierarchical data association works in two levels. In the first level, new observations (i.e., cluster of points) are matched against current dynamic or static targets. In the second level, boundary points of obstacles are updated.

3.4.3. Stereo vision based MOT

Stereo vision based methods rely on color and depth information provided by stereo pairs of images for detecting and tracking moving obstacles in the environment. Ess et al. (2010) propose a method for obstacle detection and recognition that uses only synchronized video from a forward-looking stereo camera. The focus of their work is obstacle tracking based on the per-frame output of pedestrian and car detectors. For obstacle detection, they employ a Support Vector Machine (SVM) classifier with Histogram of Oriented Gradients (HOG) features for categorizing each image region as obstacle or non-obstacle. For obstacle tracking, they apply a hypothesize-and-verify strategy for fitting a set of trajectories to the potentially detected obstacles, such that these trajectories together have a high posterior probability. The set of candidate trajectories is generated by Extended Kalman Filters (EKFs) initialized with obstacle detections. Finally, a model selection technique is used to retain only a minimal and conflict-free set of trajectories that explain past and present observations. Ziegler, Bender, Schreiber et al. (2014) describe the architecture of the modified Mercedes-Benz S-Class S500, Bertha, which drove autonomously on the historic Bertha-Benz-Memorial-Route. For MOT, dense disparity images are reconstructed from stereo image pairs using Semi-Global Matching (SGM). All obstacles within the 3D environment are approximated by sets of thin and vertically oriented rectangles called super-pixels or stixels. Stixels are tracked over time using a Kalman filter. Finally, stixels are segmented into static background and moving obstacles using spatial, shape, and motion constraints. The spatio-temporal analysis is complemented by an appearance-based detection and recognition scheme, which exploits category-specific (pedestrian and vehicle) models and increases the robustness of the visual perception. The real-time recognition consists of three main phases: Region Of Interest (ROI) generation, obstacle classification, and object tracking. Chen et al. (2017) compute a disparity map from a stereo image pair using a semi-global matching algorithm. Assisted by disparity maps, boundaries in the image segmentation produced by simple linear iterative clustering are classified into coplanar, hinge, and occlusion. Moving points are obtained during ego-motion estimation by a modified RANdom SAmple Consensus (RANSAC) algorithm. Finally, moving obstacles are extracted by merging super-pixels according to boundary types and their movements.

3.4.4. Grid map based MOT

Grid map based methods start by constructing an occupancy grid map of the dynamic environment (Petrovskaya et al., 2012). The map construction step is followed by data segmentation, data association, and filtering steps in order to provide object level representation of the scene. Nguyen et al. (2011) propose a grid-based method for detection and tracking of moving objects using stereo camera. The focus of their work is pedestrian detection and tracking. 3D points are reconstructed from a stereo image pair. An inverse sensor model is used to estimate the occupancy probability of each cell of the grid map based on the associated 3D points. A hierarchical segmentation method is employed to cluster grid cells into segments based on the regional distance between cells. Finally, an Interactive Multiple Model (IMM) method is applied to track moving obstacles. Azim and Aycard (2014) use an octree-based 3D local occupancy grid map that divides the environment into occupied, free, and unknown voxels. After construction of the local grid map, moving obstacles can be detected based on inconsistencies between observed free and occupied spaces in the local grid map. Dynamic voxels are clustered into moving objects, which are further divided into layers. Moving objects are classified into known categories (pedestrians, bikes, cars, or buses) using geometric features extracted from each layer. Ge et al. (2017) leverage a 2.5D occupancy grid map to model static background and detect moving obstacles. A grid cell stores the average height of 3D points whose 2D projection falls into the cell space domain. Motion hypotheses are detected from discrepancies between the current grid and the background model.

3.4.5. Sensor fusion based MOT

Sensor fusion-based methods fuse data from various kinds of sensors (e.g., LIDAR, RADAR, and camera) in order to explore their individual characteristics and improve environment perception. Darms et al. (2009) present the sensor fusion-based method for detection and tracking of moving vehicles adopted by the self-driving car Boss (Urmson et al., 2008) (Carnegie Mellon University's car that finished in first place in the 2007 DARPA Urban Challenge). The MOT subsystem is divided into two layers. The sensor layer extracts features from sensor data that may be used to describe a moving obstacle hypothesis according to either a point model or a box model. The sensor layer also attempts to associate features with currently predicted hypotheses from the fusion layer. Features that cannot be associated to an existing hypothesis are used to generate new proposals. An observation is

generated for each feature associated with a given hypothesis, encapsulating all information that is necessary to update the estimation of the hypothesis state. Based on proposals and observations provided by the sensor layer, the fusion layer selects the best tracking model for each hypothesis and estimates (or updates the estimation of) the hypothesis state using a Kalman Filter. Cho et al. (2014) describe the new MOT subsystem used by the new experimental self-driving car of the Carnegie Mellon University. The previous MOT subsystem, presented by Darms et al. (2009), was extended for exploiting camera data, in order to identify categories of moving objects (e.g., car, pedestrian, and bicyclists) and to enhance measurements from automotive-grade active sensors, such as LIDARs and RADARs. Mertz et al. (2013) use scan lines that can be directly obtained from 2D LIDARs, from the projection of 3D LIDARs onto a 2D plane, or from the fusion of multiple sensors (LIDAR, RADAR and camera). Scan lines are transformed into world coordinates and segmented. Line and corner features are extracted for each segment. Segments are associated with existing obstacles and kinematics of objects are updated using a Kalman filter. Na et al. (2015) merge tracks of moving obstacles generated from multiple sensors, such as RADARs, 2D LIDARs, and a 3D LIDAR. 2D LIDAR data is projected onto a 2D plane and moving obstacles are tracked using Joint Probabilistic Data Association Filter (JPDAF). 3D LIDAR data is projected onto an image and partitioned into moving obstacles using a region growing algorithm. Finally, poses of tracks are estimated or updated using Iterative Closest Points (ICP) matching or image-based data association. Xu et al. (2015) describe the context-aware tracking of moving obstacles for distance keeping used by the new experimental self-driving car of the Carnegie Mellon University. Given the behavioral context, a ROI is generated in the road network. Candidate targets inside the ROI are found and projected into road coordinates. The distance-keeping target is obtained by associating all candidate targets from different sensors (LIDAR, RADAR, and camera). Xue et al. (2017) fuse LIDAR and camera data to improve the accuracy of pedestrian detection. They use prior knowledge of a pedestrian height to reduce false detections. They estimate the height of the pedestrian according to pinhole camera equation, which combines camera and LIDAR measurements.

3.4.6. Deep learning based MOT

Deep learning based methods use deep neural networks for detecting positions and geometries of moving obstacles, and tracking their future states based on current camera data. Huval et al. (2015) propose a neural-based method for detection of moving vehicles using the Overfeat Convolutional Neural Network (CNN) (Sermanet et al., 2013) and monocular input images with focus on real-time performance. The Overfeat CNN aims at predicting location and range distance (depth) of cars in the same driving direction of the ego-vehicle using only the rear view of them. Mutz et al. (2017) address moving obstacle tracking for a closely related application known as "follow the leader", which is relevant mainly for convoys of self-driving cars. The tracking method is built on top of the Generic Object Tracking Using Regression Networks (GOTURN) (Held et al., 2016). GOTURN is a pre-trained deep neural network capable of tracking generic objects without further training or object-specific fine-tuning. Initially, GOTURN receives as input an image and a manually delimited bounding box of the leader vehicle. It is assumed that the object of interest is in the center of the bounding box. Subsequently, for every new image, GOTURN gives as output an estimate of the position and geometry (height and width) of the bounding box. The leader vehicle position is estimated using LIDAR points that fall inside the bounding box and are considered to be vehicle.

3.5. Traffic Signalization Detection

The **Traffic Signalization Detector TSD** subsystem is responsible for detecting and recognizing signs defined in the traffic rules so that the car can take correct decisions according to the traffic law. There are many tasks related to traffic signalization, and in this review, we explore three main topics: traffic lights, traffic signs, and pavement markings in the environment around the self-driving car. Each of these topics are described in detail in the next subsections.

3.5.1. Traffic light detection and recognition

Traffic light detection and recognition involve detecting the position of one or more traffic lights in the environment around the car (e.g., represented in an image) and recognizing their states (red, green, and yellow). Various methods for traffic light detection and recognition have been proposed in the literature. Here, we review only the most recent and relevant ones. For a more comprehensive review, readers are referred to Jensen et al. (2016).

Methods for traffic light detection and recognition can be mainly categorized into two classes: model-based and learning-based. Traffic lights have a well-defined structure in terms of color and shape information. A common traffic light has three bulbs (one for each state: red, green and yellow) and a well-defined form. Therefore, earlier, most of the approaches for traffic light detection and recognition were model-based. These approaches relied on hand-crafted feature engineering, which tried to leverage information humans have about the color and shape of the object to build a model capable of detecting and/or recognizing it. Methods that used color (Diaz-Cabrera et al., 2015, 2012) and shape (Omachi & Omachi, 2010; Sooksatra & Kondo, 2014; Trehard et al., 2014) information were not robust when assumptions were not strictly observed. To increase their robustness, a combination of different features (e.g., color, shape, and structure) was proposed (Gomez et al., 2014; Koukoumidis et al., 2011; Zhang et al., 2014). For example, Zhang et al. (2014) proposed a multi-feature system that combines both color (using color segmentation), shape/structure (using black box detection), and geographic information (by only using the system when known traffic lights are expected). Their system, however, suffer from the high number of hyper-parameters common on model-based approaches, which usually implicates the need of recalibration under certain circumstances. The authors performed the experiments on an in-house private data set and stated that failures were due to over-exposure, occlusions, non-standard installation of traffic lights, and several other situations that are not unusual in real-world cases. This combination, in the context of model-based approaches, showed not to be sufficient. Therefore, researchers began to introduce learning-based approaches.

In learning-based approaches, features were still hand-crafted, but detection and/or recognition processes were changed from rule-based to learning-based. Cascade classifiers (Lindner et al., 2004) were probably the first attempt to learning-based approaches. Eventually, popular combinations of HoG and Gabor features with classifiers (such as SVM (Jang et al., 2014), AdaBoost (Gong et al., 2010), and JointBoost (Haltakov et al., 2015)) were also investigated. More recently, end-to-end methods (i.e., without the need of hand-crafted features) outperformed most of the model-based ones. Fernandes et al. (2014) employed GPS data and a traffic light location database to identify a region of interest in the image, and a Convolutional Neural Network (CNN) to recognize the traffic light state. Furthermore, state-of-the-art general object detectors (Liu et al., 2016; Redmon & Farhadi, 2017; Ren et al., 2015) were successfully applied to the detection of traffic lights (often without recognizing their states). These general-purpose deep object detectors (or simply deep detectors, as they are often called), comprehensively, do not provide a breakdown on the performance of the traffic light detection and recognition task. Even though, unlike the model-based approaches, these deep detectors tend to be more robust to over-exposure, color distortions, occlusions, and others. A more complete discussion of these deep detectors applied to the traffic light detection can be found in Jensen et al. (2017). There, the authors apply the YOLO (Redmon & Farhadi, 2017) on the LISA (Jensen et al., 2016) dataset and achieve 90.49% AUC when using LISA's training set. However, the performance drops to 58.3% when

using training data from another dataset. Despite of the fact, it is still an improvement over previous methods, and it demonstrates that there is still a lot to be done. Learning-based approaches, especially those using deep learning, require large amounts of annotated data. Only recently large databases with annotated traffic lights are being made publicly available, enabling and powering learning-based approaches (Behrendt et al., 2017; Jensen et al., 2016; Yu et al., 2018).

Despite of the advances on traffic light detection and recognition research, little is known about what is being used by research self-driving cars. Perhaps, one of the main reasons for this is that there were no traffic lights in the 2007 DARPA Urban Challenge. First place and second place finishers of this challenge (the Carnegie Mellon University's team with their car Boss (Urmson et al., 2008) and the Stanford University's team with their car Junior (Montemerlo et al., 2008), respectively) recognized that traffic lights contribute to the complexity of urban environments and that they were unable to handle them at that time. In 2010, the Stadtpilot project's presented their autonomous vehicle Leonie on public roads of Braunschweig, Germany (Nothdurft et al., 2011). Leonie used information about traffic light positions from a map and Car-to-X (C2X) communication to recognize traffic light states. However, during demonstrations a co-driver had to enter the traffic light state when C2X was not available. In 2013, the Carnegie Mellon University tested their self-driving car on public roads for over a year (Wei et al., 2013). The Carnegie Mellon's car used cameras to detect traffic lights and employed vehicle-to-infrastructure (V2I) communication to retrieve information from DSRC-equipped traffic lights. In 2013, Mercedes-Benz tested their robotic car Bertha (Ziegler, Bender, Schreiber et al., 2014) on the historic Bertha-Benz-Memorial-Route in Germany. Bertha used vision sensors and prior (manual) information to detect traffic lights and recognize their states (Lindner et al., 2004). However, they state that the recognition rate needs to be improved for traffic lights at distances above 50 m.

3.5.2. Traffic sign detection and recognition

Traffic sign detection and recognition involve detecting the locations of traffic signs in the environment and recognizing their categories (e.g., speed limit, stop, and yield sign.). For reviews on methods for traffic sign detection and recognition, readers are referred to Gudigar et al. (2016) and Mogelmose et al. (2012).

Earlier, most of the approaches for traffic sign detection and recognition were model-based (Barnes et al., 2008; Gao et al., 2006) and used simple features (e.g., colors, shapes, and edges). Later, learning-based approaches (such as SVM Lafuente-Arroyo et al., 2010, cascade classifiers Pettersson et al., 2008, and LogitBoost Overett & Petersson, 2011) started leveraging simple features, but evolved into using more complex ones (e.g., patterns, appearance, and templates). However, these approaches commonly tended to not generalize well. In addition, they usually needed fine-tuning of several hyper-parameters. Furthermore, some methods worked only with recognition and not with detection, perhaps because of the lack of data. Only after large databases were made available (such as the well-known German Traffic Sign Recognition (GTSRB) (Stallkamp et al., 2012) and Detection (GTSDB) (Houben et al., 2013) Benchmarks, with 51,839 and 900 frames, respectively) that learning-based approaches (Houben et al., 2013; Mathias et al., 2013) could finally show their power, although some of them were able to cope with fewer examples (De Souza et al., 2013). With the release of even larger databases (such as STSD (Larsson & Felsberg, 2011) with over 20,000 frames, LISA (Jensen et al., 2016) with 6610 frames, BTS (Mathias et al., 2013) 25,634 frames for detection and 7125 frames for classification, and Tsinghua-Tencent 100k (Zhu et al., 2016) with 100,000 frames), learning-based approaches improved and achieved far better results when compared to their model-based counterparts. Some of these datasets report the number of frames including frames with background only. Following the rise of deep learning in general computer vision tasks, convolutional neural networks (Zhu et al., 2016) are the state-of-the-art for traffic sign detection and recognition, achieving up to 99.71% and 98.86% of F1 score on the recognition task for the GTSRB and BTS respectively.

Once more, little can be said about what is being employed for traffic sign detection and recognition by research self-driving cars. Again, maybe, one of the main drivers behind this is that only the stop-sign had to be detected and recognized in the 2007 DARPA Urban Challenge, since the map had detailed information on speed limits and intersection handling (Thrun, 2010). Some researchers (such as those of Bertha (Ziegler, Bender, Schreiber et al., 2014)) still prefer to rely on annotations about speed limit, right-of-way, among other signs. Other researchers (Fernandes et al., 2014) stated that their self-driving cars can handle traffic signs but did not provide information about their method.

3.5.3. Pavement marking detection and recognition

Pavement marking detection and recognition involve detecting the positions of pavement marking and recognizing their types (e.g., lane markings, road markings, messages, and crosswalks). Most researches deal with only one type of pavement marking at a time and not with all of them at the same time. This may happen because there is neither a widely used database nor a consensus on which set of symbols researchers should be focused on when dealing with pavement marking detection and recognition.

One important pavement marking is the lane definition in the road. Earlier, most of the methods for lane marking detection were model- or learning-based (McCall & Trivedi, 2006). Shape and color were the most usual features, and straight and curved lines (e.g., parabolic Jung & Kelber, 2005 and splines Berriel, de Aguiar et al., 2017; Berriel et al., 2015) were the most common lane representations. Berriel, de Aguiar et al. (2017) propose a complete system for performing ego-lane analysis. Among the features of the systems, the authors claim to be able of detecting lanes and their attributes, crosswalks, lane changing events, and some pavement markings. The authors also release datasets for evaluating these types of systems. Deep learning is another popular method that has gained popularity lately and approaches like (Gurghian et al., 2016) have become showed very good results. Gurghian et al. (2016) propose (i) to use two laterally-mounted down-facing cameras and (ii) to model the lateral distance estimation as a classification problem in which they employ a CNN to tackle the task. In this setting, they argue to achieve sub-centimeter accuracy with less than 2 pixels of Mean Absolute Error (MAE) in a private database. For a review on this type of methods, readers are referred to Hillel et al. (2014).

Many of the methods for lane marking detection were also attempted for road marking detection. They commonly use geometric and photometric features (Wu & Ranganathan, 2012). Furthermore, various methods for road marking detection and recognition use Inverse Perspective Mapping (IPM), which reduces perspective effect and, consequently, makes the problem easier to solve and improves the accuracy of results. More recently, several methods (Ahmad et al., 2017; Bailo et al., 2017; Lee et al., 2017) employed Maximally Stable Extremal Regions (MSER) for detecting regions of interest (i.e., regions that are likely to contain a road marking) and convolutional networks for recognizing road markings. Bailo et al. (2017) propose the combination of IPM, MSER, and DBSCAN-based algorithm to perform the detection of the road markings and the combination of PCANet with either SVM or linear regression for the classification. While they achieve up to 99.1% of accuracy when evaluating the classification task alone, it drops to 93.1% of accuracy when the performance of detection and recognition is reported together.

In the context of road markings, messages are often handled separately. Some methods for message detection and recognition (Ahmad et al., 2017) treat different messages as different categories (i.e., they firstly detect positions of messages in the scene and then recognize their categories), while most of the methods firstly recognize letters and then writings using OCR-based approaches (Greenhalgh & Mirmehdi, 2015;

Hyeon et al., 2016). The former is usually more robust to weather and lighting conditions, but the latter can recognize unseen messages.

Still in the setting of road markings, pedestrian crossings are often investigated separately. Most of the methods for crosswalk detection exploit the regular shape and black-and-white pattern that crosswalks usually present (Foucher et al., 2011; Ivanchenko et al., 2008). Therefore, in many practical applications, this task is set aside in favor of robust pedestrian detectors. For a review on these methods, readers are referred to Berriel, Rossi et al. (2017). Together with the review, Berriel, Rossi et al. (2017) present a deep learning-based system to detect the presence of crosswalks in images. The authors provide pre-trained models that can be directly applied for this task.

4. Self-driving cars' decision making

In this section, we survey relevant techniques reported in the literature for the decision-making system of self-driving cars, comprising the route planning, behavior selection, motion planning, and control subsystems.

4.1. Route planning

The **Route Planner** subsystem is responsible for computing a route, W, through a road network, from the self-driving car's initial position to the final position defined by a user operator. A Route is a sequence of way points, i.e. $W = \{w_1, w_2, \ldots, w_{|W|}\}$, where each way point, w_i, is a coordinate pair, i.e. $w_i = (x_i, y_i)$, in the Offline Maps. If the road network is represented by a weighted directed graph, whose vertices are way points, edges connect pairs of way points, and edge weights denote the cost of traversing a road segment defined by two way points, then the problem of computing a route can be reduced to the problem of finding the shortest path in a weighted directed graph. However, for large road networks, time complexity of classical shortest path algorithms, such as Dijkstra (Dijkstra, 1959) and A$*$ (Hart et al., 1968), make them impractical.

In the last decade, there have been significant advances in the performance of algorithms for route planning in road networks. Newly developed algorithms can compute driving directions in milliseconds or less, even at continental scales. For a review on algorithms for route planning in road networks that are applicable to self-driving cars, readers are referred to Bast et al. (2015).

Techniques for route planning in road networks provide different trade-offs in terms of query time, preprocessing time, space usage, and robustness to input changes, among other factors. Such techniques can be categorized into four classes (Bast et al., 2015): (i) goal-directed, (ii) separator-based, (iii) hierarchical, and (iv) bounded-hop. These classes may also be combined.

4.1.1. Goal-directed techniques

Goal-directed techniques guide the search from the source vertex to the target vertex by avoiding scans of vertices that are not in the direction of the target vertex. A$*$ is a classic goal-directed shortest path algorithm. It achieves better performance than the Dijkstra's algorithm by using a lower-bound distance function on each vertex, which causes vertices that are closer to the target to be scanned earlier. A$*$, Landmarks, and Triangle inequality (ALT) algorithm (Goldberg & Harrelson, 2005) enhances A$*$ by picking a small set of vertices as landmarks. During the preprocessing phase, distances between all landmarks and all vertices are computed. During the query phase, a valid lower-bound distance for any vertex is estimated, using triangle inequalities involving landmarks. The query performance and correctness depend on the wise choice of vertices as landmarks.

Another goal-directed algorithm is Arc Flags (Hilger et al., 2009). During the preprocessing phase, the graph is partitioned into cells with a small number of boundary vertices and a balanced (i.e., similar) number of vertices. Arc flags for a cell i are computed by growing backwards a shortest path tree from each boundary vertex, setting the i-th flag for all arcs (i.e., edges) of the tree. During the query phase, the algorithm prunes arcs that do not have the flag set for the cell that contains the target vertex. The arc flags method has high preprocessing times, but the fastest query times among goal-directed techniques.

4.1.2. Separator-based techniques

Separator-based techniques are based on either vertex or arc separators. A vertex (or arc) separator is a small subset of vertices (or arcs) whose removal decomposes the graph into several balanced cells. The vertex separator-based algorithm uses vertex separators to compute an overlay graph. Shortcut arcs are added to the overlay graph such that distances between any pair of vertices from the full graph are preserved. The overlay graph is much smaller than the full graph and is used to accelerate the query algorithm. The High-Performance Multilevel Routing (HPML) algorithm (Delling et al., 2009) is a variant of this approach that significantly reduces query time, but at the cost of increasing space usage and preprocessing time, by adding many more shortcuts to the graph across different levels.

The arc separator-based algorithm uses arc separators to decompose the graph into balanced cells, attempting to minimize the number of cut arcs, which connect boundary vertices of different cells. Shortcuts are added to the overlay graph in order to preserve distances between boundary vertices within each cell. Customizable Route Planning (CRP) algorithm (Delling et al., 2015) is a variant of this approach, which was designed to meet requirements of real-world road networks, such as handling turn costs and performing fast updates of the cost function. Its preprocessing has two phases. The first phase computes the multilevel partition and the topology of the overlays. The second phase computes the costs of clique arcs by processing the cells bottom-up and in parallel. Queries are processed as bidirectional searches in the overlay graph.

4.1.3. Hierarchical techniques

Hierarchical techniques exploit the inherent hierarchy of road networks, in which main roads such as highways compound a small arterial subnetwork. When the source and target vertices are distant, the query algorithm only scans vertices of the subnetwork. The preprocessing phase computes the importance of vertices or arcs according to the actual shortest path structure. The Contraction Hierarchies (CH) algorithm (Geisberger et al., 2012) is a hierarchical technique that implements the idea of creating shortcuts to skip vertices with low importance. It repeatedly executes vertex contraction operations, which remove from the graph the least important vertex and create shortcuts between each pair of neighboring vertices, if the shortest path between them is unique and contains the vertex to be removed (i.e., contracted). CH is versatile, thus serving as a building block for other point-to-point algorithms and extended queries.

The REACH algorithm (Gutman, 2004) is a hierarchical technique that, during the preprocessing phase, computes centrality measures (reach values) on vertices and, during query phase, uses them to prune Dijkstra-based bidirectional searches. Let P be a shortest path that contains the vertex w_v, which goes from the source vertex w_s to the target vertex w_t. The reach value of w_v with respect to P is $r(w_v, P) = \min\{\text{distance}(w_s, w_v), \text{distance}(w_v, w_t)\}$.

4.1.4. Bounded-hop techniques

Bounded-hop techniques precompute distances between pairs of vertices by adding virtual shortcuts to the graph. Since precomputing distances among all pairs of vertices is prohibitive for large networks, bounded-hop techniques aim to get the length of any virtual path with very few hops. Hub Labeling (HL) (Cohen et al., 2003) is a bounded-hop algorithm that, during the preprocessing phase, computes a label $L(w_u)$ for each vertex w_u of the graph, which consists of a set of hub vertices of w_u and their distances from w_u. These labels are chosen such that they obey the cover property: for any pair (w_s, w_t) of vertices, the intersection of labels $L(w_s)$ and $L(w_t)$ must contain at least one vertex

of the shortest path from w_s to w_t. During the query phase, the distance (w_s, w_t) can be determined in linear time by evaluating the distances between hub vertices present in the intersection of labels $L(w_s)$ and $L(w_t)$. HL has the fastest known queries in road networks, at the cost of high space usage.

The HL-∞ algorithm (Abraham et al., 2012) exploited the relationship between hub labelings and vertex orderings. It uses preprocessing algorithms for computing orderings that yield small labels. The iterative range optimization algorithm for vertex ordering (Abraham et al., 2012) makes the query time of HL-∞ algorithm twice as fast as HL. It starts with some vertex ordering (e.g., the one given by CH) and proceeds in a given number of iterative steps, each reordering a different range of vertices in decreasing order of importance. The Hub Label Compression (HLC) algorithm (Delling, Goldberg and Werneck, 2013) reduces space usage by an order of magnitude, at the cost of higher query times, by combining common substructures that appear in multiple labels.

Another bounded-hop algorithm is Transit Node Routing (TNR) (Arz et al., 2013), which uses distance tables on a subset of the vertices. During the preprocessing phase, it selects a small set of vertices as transit nodes and computes all pairwise distances between them. From transit nodes, for each vertex w_u, it computes a set of access nodes. If there is a shortest path from w_u, such that w_v is the first transit node in it, then the transit node w_v is an access node of w_u. It also computes distances between each vertex and its access nodes. A natural approach for selecting the transit node set is to select vertex separators or boundary vertices of arc separators as transit nodes. During query phase, the distance table is used to select the path from the source vertex w_s to the target vertex w_t that minimizes the combined distance w_s –$a(w_s)$–$a(w_t)$ –w_t, where $a(w_s)$ and $a(w_t)$ are access nodes. If the shortest path does not contain a transit node, then a local query is performed (typically CH).

4.1.5. Combinations

Individual techniques can be combined into hybrid algorithms that exploit different graph properties. The REAL algorithm (Goldberg et al., 2006) combines REACH and ALT. The ReachFlags algorithm (Bauer et al., 2010) combines REACH and Arc Flags. The SHARC algorithm (Bauer & Delling, 2008) combines the computation of shortcuts with multilevel Arc Flags. The CHASE algorithm (Bauer et al., 2010) combines CH with Arc Flags. The TNR+AF algorithm (Bauer et al., 2010) combines TNR and Arc Flags. The PHAST algorithm (Delling, Goldberg, Nowatzyk et al., 2013) can be combined with several techniques in order to accelerate them by exploiting parallelism of multiple core CPUs and GPUs.

Bast et al. (2015) evaluated experimentally many of the route planning techniques described here, using the well-known continent-sized benchmark Western Europe and a simplified model of real-world road networks. Table 1 shows the results of their experimental analysis. For each technique, Table 1 presents the total memory space usage, total preprocessing time, number of vertices scanned by a query on average, and average query time.

Table 1
Performance of route planning techniques on Western Europe (Bast et al., 2015).

Algorithm	Data structures		Queries	
	Space (GB)	Time (h:m)	Scanned vertices	Time (μs)
Dijstra	0.4	–	9326,696	2,195,080
Bidirect. Dijstra	0.4	–	4914,804	1,205,660
CRP	0.9	1:00	2766	1650
Arc flags	0.6	0:20	2646	408
CH	0.4	0:05	280	110
CHASE	0.6	0:30	28	5.76
HLC	1.8	0:50	–	2.55
TNR	2.5	0.22	–	2.09
TNR+AF	5.4	1:24	–	0.70
HL	18.8	0.37	–	0.56
HL-∞	17.7	60:00	–	0.25
Table lookup	1,208,358	145:30	–	0.06

4.2. Path planning

The **Path Planner** subsystem computes a set of Paths, $= \{P_1, P_2, \ldots, P_{|P|}\}$, considering the current route, the self-driving car's State, the internal representation of the environment, as well as traffic rules. A Path $P_j = \{p_1, p_2, \ldots, p_{|P|}\}$ is a sequence of poses $p_i = (x_i, y_i, \theta_i)$, which are car positions and respective orientations in the Offline Maps. The first poses of the paths of P is the current self-driving car position and orientation, and these paths extend some tens or hundreds of meters away from the self-driving car current position.

Various methods for path planning have been proposed in the literature. We review those that were evaluated experimentally using real-world self-driving cars. For more comprehensive reviews on these methods readers are referred to González et al. (2015) and Paden et al. (2016).

Methods for path planning can be mainly categorized into two classes: graph search based and interpolating curve based (González et al., 2015; Paden et al., 2016).

4.2.1. Graph search based techniques

Graph search based techniques searches for the best paths between the car's current state and a goal state in a state space represented as a graph. The goal state is a pose near a way point w_i of the current route W. These techniques discretize the search space imposing a graph on an occupancy grid map with centers of cells acting as neighbors in the search graph. The most common graph search based techniques for path planning of self-driving cars are Dijkstra, A-star, and A-star variants.

The Dijkstra algorithm (Dijkstra, 1959) finds the shortest path between the initial node and goal node of a graph. Dijkstra repeatedly examines the closest not yet examined node, adding its neighbors to the set of nodes to be examined, and stops when the goal node is attained. Arnay et al. (2016) used the Dijkstra algorithm to compute a path (and also a route; see Section 4.1) for the self-driving car Verdino. Bacha et al. (2008) employ Dijkstra to construct a path for the self-driving car Odin to navigate toward and reversing out of a parking spot. Kala and Warwick (2013) used Dijkstra to generate a path (and also a route), which were tested only in computer simulations.

The A-star algorithm (Hart et al., 1968) is an extension of Dijkstra, which performs fast graph search by assigning weights to nodes based on a heuristic-estimated cost to the goal node. Leedy et al. (2007) employed the A-star algorithm to build a path for the self-driving car Rocky, which participated in the 2005 DARPA Grand Challenge. Ziegler et al. (2008) proposed a path planning that combines A-star with two different heuristic cost functions, namely Rotation Translation Rotation (RTR) metric and Voronoj graph, to compute a path. The first one accounts for kinematics constraints of the car, while the second one incorporates knowledge of shapes and positions of obstacles. The method was tested in the robotic car AnnieWAY, which participated of the 2007 DARPA Urban Challenge.

Other authors propose variations of the A-star algorithm for path planning. Urmson et al. (2008) proposed the anytime D-star to compute a path for the self-driving car Boss (Carnegie Mellon University's car that claimed first place in the 2007 DARPA Urban Challenge). Dolgov et al. (2010) proposed the hybrid-state A-star to compute a path for the robotic car Junior (Montemerlo et al., 2008) (Stanford University's car that finished in second place in the 2007 DARPA Urban Challenge). Both anytime D-star and hybrid-state A-star algorithms merge two heuristics – one non-holonomic, which disregards obstacles, and the other holonomic, which considers obstacles – and were used for path planning in an unstructured environment (parking lot). Chu et al. (2015) proposed a variation of A-star to build a path that considers

car's kinematic constraints, which ignores the resolution of grid cells and creates a smooth path. Yoon et al. (2015) proposed a variation of A-star to compute a path that accounts for kinematic constraints of the autonomous vehicle Kaist.

4.2.2. Interpolating curve based techniques

Interpolating curve based techniques take a previously known set of points (e.g., waypoints of a route) and generate a new set of points that depicts a smooth path. The most usual interpolating curve based techniques for path planning of self-driving cars are spline curves.

A spline is a piecewise polynomial parametric curve divided in sub-intervals that can be defined as polynomial curves. The junction between each sub-segment is called knot (or control points), which commonly possess a high degree of smoothness constraint. This kind of curve has low computational cost, because its behavior is defined by the knots. However, its result might not be optimal, because it focuses more on achieving continuity between the parts instead of meeting road's constraints (González et al., 2015). Chu et al. (2012) and Hu et al. (2018) use cubic spline curves for path planning. Both of them construct a center line from a route extracted from a road network. They generate a series of parametric cubic splines, which represent possible path candidates, using the arc length and offset to the center line. The optimal path is selected based on a function that takes into account the safety and confort.

4.3. Behavior selection

The **Behavior Selector** subsystem is responsible for choosing the current driving behavior, such as lane keeping, intersection handling, traffic light handling, etc., by selecting a Path, P_j, in P, a pose, p_g, in P_j, and the desired velocity at this pose. The pair p_g and associated velocity is called a $Goal_g = (p_g, v_g)$. The estimated time between the current state and the $Goal_g$ is the *decision horizon*. The Behavior Selector chooses a Goal considering the current driving behavior and avoiding collisions with static and moving obstacles in the environment within the decision horizon time frame.

The Behavior Selector raises complex discussions about ethical decisions. What the system will choose to do in an inevitable accident? The priority is the passenger's safety or pedestrian's safety? These questions are not covered here because, in the literature, the approaches for real world self-driving cars did not achieve the autonomy level to consider these problems. Nevertheless, there is some research on the subject. Readers can refer to Awad et al. (2018), Conitzer et al. (2017) and Greene et al. (2016).

A self-driving car has to deal with various road and urban traffic situations, and some of them simultaneously (e.g. yield and merge intersections). Regarding that, in the literature the behavior selection problem can be divided according to the different traffic scenarios in order to solve them in parts. At DARPA Urban Challenge (Buehler et al., 2009) the principal methods used for different driving scenarios were combination of heuristics (Urmson et al., 2008), decision trees (Miller et al., 2008), and Finite State Machines (FSM) (Montemerlo et al., 2008). These methods perform well in limited, simple scenarios. Moreover, State Machines based methods have been improved and fused with other methods to cope with a larger variety of real urban traffic scenario (Aeberhard et al., 2015; Jo et al., 2015; Okumura et al., 2016; Ziegler, Bender, Schreiber et al., 2014). Ontologies can also be used as a tool to model traffic scenarios (Zhao et al., 2017, 2015). Several approaches for behavior selection consider uncertainty in the intentions and trajectories of moving objects. For that, they use probabilistic methods, such as Markov Decision Processes (MDPs) (Brechtel et al., 2011), and Partially-Observable Markov decision process (POMDP) (Brechtel et al., 2014; Galceran et al., 2017; Ulbrich & Maurer, 2013).

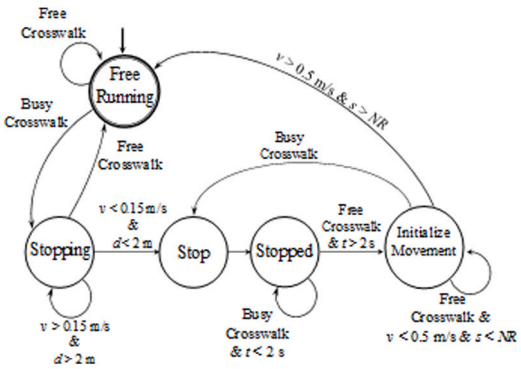


Fig. 7. IARA's behavior selector state machine for handling pedestrians in crosswalks (Guidolini et al., 2018). At the image, v is the car's velocity, d is the distance between the car and the crosswalk stop line, t is the time while the crosswalk is free, s is the car's current position, and NR is the non-return point that is after the crosswalk stop line.

4.3.1. FSM-based techniques

In FSM methods, a rule-based decision process is applied to choose actions under different traffic scenarios. The behavior is represented by states and transitions are based on discrete rules stem from perception information. The current state defines the car's present behavior. The major drawback of this method is the difficulty to model the uncertainties and complex urban traffic scenarios. One of the successful uses of FSM for urban road traffic was that employed by the Junior team in DARPA Urban Challenge (Buehler et al., 2009). They used a FSM that had states for several urban traffic scenarios. However, that competition presented a limited set of urban scenarios, if compared with real world traffic, i.e. fewer traffic participants, no traffic lights and controlled traffic jams. In a more complex scenario, the vehicle A1 (Jo et al., 2015) used a FSM for selecting a driving strategy among predefined rules so as to follow traffic regulations (e.g., lane keeping, obeying traffic lights, and keeping the car under speed limits).

In the IARA's behavior selector subsystem, a FSM is used for each scenario. It receives as input: the map; the car's current state; the current path; the perception system information, such as traffic light state (Possatti et al., 2019), moving objects detections, pedestrians in crosswalk (Guidolini et al., 2018; Sarcinelli et al., 2019), etc.; and a set of map annotations, such as speed bumps, security barriers, crosswalks and speed limits. For each scenario, a finite state machine checks some state transition rules and defines the appropriated next state. Then, considering the current state and using several heuristics, it defines a goal state in the path within the decision horizon and adjusts the goal state's velocity in order to make the car behave properly according to the scenario, i.e. stopping on red traffic lights, reducing the velocity or stopping in busy crosswalks, etc. Fig. 7 shows the IARA's FSM for handling pedestrians in crosswalks (Guidolini et al., 2018).

Aeberhard et al. (2015) modeled the behavior selection process as a network of hybrid deterministic automata and decision-trees. In order to reduce the complexity of the model, the driving task is divided into a finite set of lateral and longitudinal guidance states. The lateral guidance states consist of lane keeping and lane change states. The longitudinal guidance consists of cruise controls states and a single critical control state. The behavior selection process runs through several hierarchical levels evaluating the present situation and selecting the appropriate driving maneuver. The system was tested in a real world scenario by the BMW Group Research and Technology.

Okumura et al. (2016) combine FSM with a support vector machine (SVM) to build a classifier for a high-level behavior selector process in roundabouts situations. The decision process consists of two levels. First the SVM classifier maps the current robot state and perception data to an action, then a FSM process the action in order to output high-level commands. The SVM was trained by demonstration: a human

driver annotates the situations where the driver must stop or go through the roundabout considering the predictions of other vehicles behavior. They performed experiments in the real world with an overall success rate of 98.2%. The authors claim that this approach can also be applied in similar autonomous driving situations.

To cope with more complex scenarios, Ziegler, Bender, Schreiber et al. (2014) select the behavior using a hierarchical, concurrent state machine. This state machine was modeled using a state chart notation (Harel state chart Harel, 1987) that allows concurrent states. Their behavior selector subsystem generates a series of constraints that are derived from these concurrent state machines and used as input of a trajectory optimization problem. These constraints are formulated by the behavior selector subsystem considering information such as the characteristics of the driving corridor, the presence of static and moving objects, and the yield and merge rules. This approach was tested in a real autonomous car, named Bertha, that traveled 104 km fully autonomously through urban areas and country roads.

4.3.2. Ontology-based techniques

Ontologies are frameworks for knowledge representation that can be used to model concepts and their relationships. Zhao et al. (2015) used an ontology-based Knowledge Base to model traffic regulations and sensor data in order to help self-driving cars to understand the world. To build their decision-making system, they constructed the ontology-based Knowledge Base manually. They focused on traffic situations that occur in intersections and narrow roads. The system makes decisions considering regulations such as Right-of-Way rules, and sends decisions such as “Stop, “ToLeft”, or “Give Way” to a path planning system to change the route or stop to avoid collisions. The disadvantage of this approach is the need to design an accurate world model composed of items such as mapped lanes and traffic rules at each location, which is usually done manually by humans.

In a more recent work, Zhao et al. (2017) improved on their previous work in order use only a small portion of the Knowledge Base, making it 1/20~1/10 smaller than in the previous work — this increased the performance in terms of time making the system almost 10 times faster.

4.3.3. Markov decision processes based techniques

The Partially Observable Markov Decision Process (POMDP) framework not only addresses the uncertainty in actions transition between states, but also the uncertainty in perception. This algorithm generalizes the value iteration algorithm to estimate an optimal control policy (Thrun et al., 2005). Ulbrich and Maurer (2013) apply online POMDP to behavior selection in order to perform lane changes while driving in urban environments. To reduce the complexity of the POMDP and allow real-time decision making, the proposed algorithm was divided into two steps. In the first step, the situation is evaluated by a signal processing network. This network is a graph that considers relative distances, velocities and time to collisions with objects around the vehicle and outputs whether a lane change is possible or not. With the network output, the POMDP decision making algorithm plans online only for the current belief state. This is made by searching only the belief states that are reachable from the current state. Hence, the POMDP model only needs to have eight states. The algorithm was evaluated in real world inner city traffic showing decision coherency while maintaining a high level of decision quality.

Brechtel et al. (2014) used a continuous POMDP approach to reason about potentially hidden objects and observation uncertainty, considering the interactions of road participants. They used the idea of assuming finite sets of policies to speed up planning. In the first step, a reward function aims to optimize comfort and efficiency by returning a cost for acceleration and deceleration to reach a goal area. This step only depends on the vehicle’s state and a previously defined goal. In the second step, the other traffic participants are considered by adding a higher cost for collision with other road users. The two steps costs are summed up in a reward function with a scalar value that represents the driving objective. Their approach was evaluated in a merging scenario. The policy for this scenario and all possible outcomes of it were pre-computed offline.

Galceran et al. (2017) proposed an integrated inference and behavior selection approach that models vehicle behavior and nearby vehicles as a discrete set of policies. In this work, they used a set of hand-engineering policies to in-lane and intersection driving situations. They used Bayesian changepoint detection on the history of other vehicles to infer possible future actions, estimating a distribution over potential policies that each nearby vehicle might be executing. Then, the behavior selection algorithm executes the policy with the maximum reward value by approximating the POMDP solution that evaluates the policies of the predicted vehicle through forwarding simulation. However, they assume that majority of the driving participants behave in a regular, predictable manner, following traffic rules. The experiments were carried out using an autonomous car platform.

Instead of a set of policies, Wray et al. (2017) proposed an intersection behavior selection for Autonomous Vehicles using multiple online decision-components with interacting actions (MODIA). MODIA also consider the traffic participants but model them as separate individual Markov decision processes (MDP). Each MDP maintains its own belief and proposed action to take at each time step, generating a set of estimated actions. A lexicographic executor action function (LEAF) only executes the best (in terms of preference) action from this set (e.g. stop actions have preference). Actions are simply stop, edge or go, and encode movements by assigning desired velocity and goal points along the AV’s trajectory. MODIA remains tractable by growing linearly in the number of decisions process instantiated. This method was tested in a real autonomous car in intersection scenarios and compared with an ignorant and naive baseline algorithm successfully solving the autonomous vehicle interaction in intersection scenarios.

4.4. Motion planning

The **Motion Planner** subsystem is responsible for computing a Trajectory, T, from the current self-driving car’s State to the current Goal. This trajectory must follow the Path, P_j, defined by the Behavior Selector subsystem as close as possible, while satisfying car’s kinematic and dynamic constraints, and providing safety and comfort to the passengers. There are two basic forms of defining a trajectory. It may be defined as (i) a sequence of commands, i.e. $T^c = \{c_1, c_2, \ldots, c_{|T|}\}$, where each command $c_k = (v_k, \varphi_k, \Delta t_k)$ and, for each c_k, v_k is the desired velocity at time k, φ_k is the desired steering angle at time k, and Δt_k is the duration of c_k; or it may be defined as (ii) a sequence of states $T^s = \{s_1, s_2, \ldots, s_{|T|}\}$, where each state $s_k = (p_k, t_k)$ and, for each s_k, p_k is a pose, and t_k is the absolute time in which this pose is expected to be achieved. A Trajectory takes the car from its current State to the current Goal smoothly and safely.

Several techniques for motion planning have been proposed in the literature. We review those that were designed for on-road motion planning and were evaluated experimentally using real-world self-driving cars. On-road motion planning aims at planning trajectories that follow the route, which differs from unstructured motion planning, in which there are no lanes and, thus, trajectories are far less constrained. For more comprehensive reviews on methods for motion planning readers are referred to González et al. (2015) and Paden et al. (2016).

Methods for motion planning can be mainly categorized into four classes: graph search based, sampling based, interpolating curve based, and numerical optimization based methods (González et al., 2015; Paden et al., 2016).

4.4.1. Graph search based techniques

Graph search based techniques for trajectory planning extend those for path planning (Section 4.2) in order to specify the evolution of car's state over time. The most common graph search based trajectory planning techniques for self-driving cars are state lattice, Elastic Band (EB), and A-star.

A state lattice (Pivtoraiko et al., 2009) is a search graph in which vertices denote states and edges denote paths that connect states satisfying the robot's kinematic constraints. Vertices are placed in a regular fashion, such that the same paths, although rigidly translated and rotated, can be used to build a feasible path from each of the vertices to the goal. In this way, a trajectory to the goal is likely to be represented as a sequence of edges in the graph. State lattices are able to handle several dimensions, such as position, velocity, and acceleration, and are suitable for dynamic environments. However, they have high computational cost, because they evaluate every possible solution in the graph (González et al., 2015).

McNaughton et al. (2011) propose a conformal spatiotemporal state lattice for trajectory planning. They construct the state lattice around a centerline path. They define nodes on the road at a lateral offset from the centerline and compute edges between nodes using an optimization algorithm. This optimization algorithm finds the parameters of a polynomial function that define edges connecting any pairs of nodes. They assign to each node a state vector that contains a pose, acceleration profile, and ranges of times and velocities. The acceleration profile increases trajectory diversity at a less cost than would the finer discretization of time and velocity intervals. Furthermore, the ranges of times and velocities reduce computational cost by allowing the assignment of times and velocities to the graph search phase, instead of graph construction phase. Xu et al. (2012) propose an iterative optimization that is applied to the resulting trajectory derived from the state lattice trajectory planning proposed by McNaughton et al. (2011), which reduces the planning time and improve the trajectory quality. Gu et al. (2016) propose a planning method that fuses the state lattice trajectory planning proposed by McNaughton et al. (2011) with a tactical reasoning. A set of candidate trajectories is sampled from the state lattice, from which different maneuvers are extracted. The final trajectory is obtained by selecting a maneuver (e.g., keep lane or change lane) and choosing the candidate trajectory associated with the selected maneuver. Li et al. (2015) builds a state lattice by generating candidate paths along a route using a cubic polynomial curve. A velocity profile is also computed to be assigned to poses of the generated paths. The resulting trajectories are evaluated by a cost function and the optimal one is selected.

Another graph search based technique is the elastic band, which was first used in path planning and later in trajectory. An elastic band is a search graph with elastic vertices and edges. An elastic vertex is defined by augmenting the spatial vertex with an in-edge and out-edge that connect the neighboring spatial vertices. The path is found by an optimization algorithm that balances two forces – repulsive forces generated by external obstacles and contractive forces generated by the neighboring points to remove band slackness. This technique demonstrates continuities and stability, it has non-deterministic run-time, but requires a collision-free initial path.

Gu et al. (2015) propose a decoupled space–time trajectory planning method. The trajectory planning is divided into three phases. In the first phase, a collision-free feasible path is extracted from an elastic band, considering road and obstacles constraints, using a pure-pursuit controller and a kinematic car model (see Section 4.6.2). In the second phase, a velocity profile is suggested under several constraints, namely speed limit, obstacle proximity, lateral acceleration and longitudinal acceleration. Finally, given the path and velocity profile, trajectories are computed using parametric path spirals. Trajectories are evaluated against all static and moving obstacles by simulating their future movements.

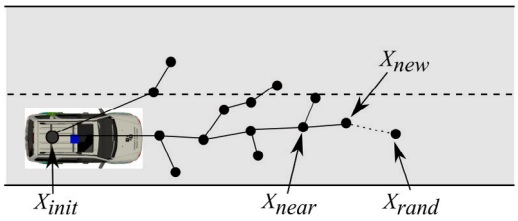


Fig. 8. Example of a trajectory generated by a RRT method, where X_{init} is the car's State, X_{rand} is the random state, X_{near} is the nearest vertex of the tree to X_{rand} and X_{new} is the new state as close as possible to X_{rand}.

The A-star algorithm is commonly used for path planning (Section 4.2). Fassbender et al. (2016) propose two novel A-star node expansion schemes for trajectory planning. The first scheme tries to find a trajectory that connects the car's current node directly to the goal node using numerical optimization. The second scheme uses a pure-pursuit controller to generate short edges (i.e., short motion primitives) that guide the car along the global reference path.

4.4.2. Sampling based techniques

Sampling based techniques randomly sample the state space looking for a connection between the car's current state and the next goal state. The most used sampling based technique for trajectory planning of self-driving cars is Rapidly-exploring Random Tree (RRT).

RRT methods for trajectory generation (LaValle & Kuffner Jr, 2001) incrementally build a search tree from the car's current state using random samples from the state space. For each random state, a control command is applied to the nearest vertex of the tree in order to create a new state as close as possible to the random state. Each vertex of the tree represents a state and each directed edge represents a command that was applied to extend a state. Candidate trajectories are evaluated according to various criteria. Fig. 8 shows an example of a trajectory generated by a RRT method from the current car's State to a random state. RRT methods have low computational cost for high-dimensional spaces and always find a solution, if there is one and it is given enough time. However, its result is not continuous and jerky (González et al., 2015).

Radaelli et al. (2014) propose a RRT method for trajectory planning of the self-driving car IARA. They present new variants to the standard RRT in order to bias the location of random states toward the path, select the most promising control commands to extend states, discard non-promising states, and choose the set of states that constitutes the best trajectory. It also reuses part of the trajectory built in the previous planning cycle. Du et al. (2016) propose a RRT method that uses the drivers' visual search behavior on roads to guide the state sampling of RRT. Drivers use both a "near point" and a "far point" to drive through curved roads. They exploit this characteristic of drivers' visual search behavior shown on curved roads to guide the RRT method. Furthermore, they adopt a post-processing, based on B-splines, to generate smooth, continuous, and feasible trajectories.

4.4.3. Interpolating curve based techniques

Interpolating curve based techniques interpolate a previously known set of poses (e.g., the path's poses) and build a smoother trajectory that considers car's kinematic and dynamic constraints, comfort, obstacles, among other parameters. The most common interpolating curve based techniques for trajectory planning of self-driving cars are clothoid curves (González et al., 2015).

A clothoid curve allows defining trajectories with linear variable curvature, so that transitions between straight to curved segments are smooth. However, a clothoid curve has high computational cost, because of the integrals that define it, and it depends on global waypoints (González et al., 2015). Alia et al. (2015) use clothoid tentacles

for trajectory planning. Tentacles are computed for different speeds and different initial steering angles, starting from the car's center of gravity and taking the form of clothoids. Tentacles are classified as navigable or not navigable using a path tracking controller that derives the steering angle commands from the tentacles' geometric parameters and running a collision-check on an occupancy grid map. Among the navigable tentacles, the best tentacle is chosen based on several criteria. Mouhagir et al. (2017, 2016) also use clothoid tentacles for trajectory planning. However, they use an approach inspired in Markov Decision Process (MDP) to choose the best tentacle.

4.4.4. Numerical optimization based techniques

Numerical optimization based techniques minimize or maximize a function with constrained variables. The most common numerical optimization based techniques for trajectory planning of self-driving cars are function optimization and model-predictive methods.

Function optimization methods find a trajectory by minimizing a cost function that considers trajectory's constraints, such as position, velocity, acceleration, and jerk. These methods can easily consider car's kinematic and dynamic constraints and environment's constraints. However, they have high computational cost, because the optimization process takes place at each motion state, and depend on global waypoints (González et al., 2015). Ziegler, Bender, Dang et al. (2014) uses a function optimization method for trajectory planning of the self-driving car Bertha. They find the optimal trajectory that minimizes a cost function and obeys trajectory's constraints. The cost function is composed of a set of terms that make the trajectory to follow the middle of the driving corridor (which is equivalent to a path) at a specified velocity, penalize strong acceleration, dampen rapid changes in acceleration, and attenuate high yaw rates.

Model predictive methods for trajectory planning (Howard & Kelly, 2007) produce dynamically feasible control commands between car's current state and next goal state by using models that predict future states. They can be used to solve the problem of generating parameterized control commands that satisfy state constraints whose dynamics can be expressed by differential equations for example. Ferguson et al. (2008) uses a model-predictive method for trajectory planning of the self-driving car Boss (Carnegie Mellon University's car that claimed first place in the 2007 DARPA Urban Challenge) (Urmson et al., 2008). They generate trajectories to a set of goal states derived from the centerline path. To compute each trajectory, they use an optimization algorithm that gradually modifies an initial approximation of the trajectory control parameters until the trajectory end point error is within an acceptable bound. The trajectory control parameters are the trajectory length and three knot points of a spline curve that defines the curvature profile. A velocity profile is generated for each trajectory based on several factors, including the velocity limit of the current road, maximum feasible velocity, and goal state velocity. The best trajectory is selected according to their proximity to obstacles, distance to the centerline path, smoothness, end point error, and velocity error.

Li et al. (2017) use a state sampling-based trajectory planning scheme that samples goal states along a route. A model-predictive path planning method is applied to produce paths that connect the car's current state to the sampled goal states. A velocity profile is used to assign a velocity to each of the states along generated paths. A cost function that considers safety and comfort is employed to select the best trajectory.

Cardoso et al. (2017) used a model-predictive method for trajectory planning of the self-driving car IARA. To compute the trajectory, they use an optimization algorithm that finds the trajectory control parameters that minimize the distance to the goal state, distance to the path, and proximity to obstacles. The trajectory control parameters are the total time of the trajectory, tt, and three knot points, k_1, k_2 and k_3, that define a cubic spline curve which specifies the steering angle profile, i.e., the evolution of the steering angle during tt. Fig. 9 shows an example of a trajectory from the current car's State (in blue) to the current Goal (in yellow), that follows the path (in white) and avoids obstacles (in shades of gray). Fig. 10 shows the steering angle profile of the trajectory illustrated in Fig. 9.

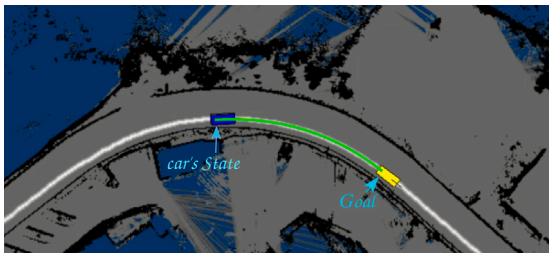


Fig. 9. Example of a trajectory from the current car's State (blue rectangle), to the Goal (yellow rectangle), which is illustrated in an excerpt of an OGM. The path is indicated in white and the trajectory in green. In the OGM, the shades of gray indicate the occupancy probability: black corresponds to occupied with high certainty and white to free with high certainty; the light blue indicates unknown or non-sensed regions.

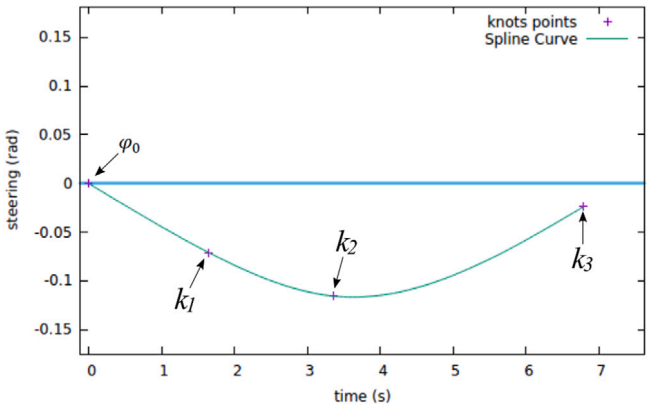


Fig. 10. Spline curve, defined by φ_0, and k_1, k_2 and k_3, that specifies the evolution of the steering angle during the total time of the trajectory, $tt = 6.7$ s. The knot point k_1 is defined at $t = tt/4$, k_2 at $t = tt/2$ and k_3 at $t = tt$.

4.5. Obstacle avoidance

The **Obstacle Avoider** subsystem receives the Trajectory computed by the Motion Planner subsystem and changes it (typically reducing the velocity), if necessary, to avoid collisions. There is no much literature on methods for performing the functions of the Obstacle Avoider subsystem.

Guidolini et al. (2016) proposes an obstacle avoider subsystem that, at each motion plan cycle, receives an online map that represents the environment around the car, the current state of the self-driving car in the online map, and the trajectory planned by the motion planner subsystem. The obstacle avoider subsystem simulates the trajectory and, if a crash is predicted to occur in the trajectory, the obstacle avoider decreases the linear velocity of the self-driving car to prevent the crash. Fig. 11 shows the block diagram of the obstacle avoider.

Cao et al. (2016) uses RADAR data to calculate the distance between a self-driving car and the obstacles in the environment. After that the time to collision (TTC) is calculated and compared with a distance threshold. If TTC exceeds the threshold the system trigger a decision-maker subsystem that decides the self-driving car behavior (lane keeping, lane changing, speeding up, adaptive following and emergency braking).

He et al. (2019) proposed a system to avoid collisions based on a hierarchical control architecture that consists of a decision making layer, which contains a dynamic threat assessment model that continuously analyzes the risk associated with collision, and a path planner that calculates a collision-free path that considers the kinematics and dynamics constraints of the self-driving car when an emergency situation happens.

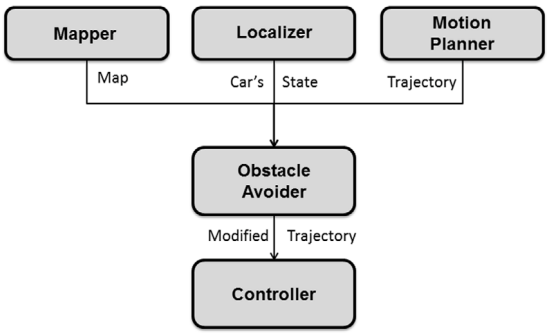


Fig. 11. Block diagram of the obstacle avoider system proposed by Guidolini et al. (2016). The system receives as input the online map, current car's state and the trajectory and simulates the trajectory. If any crash occurs, the obstacle avoider system decreases the linear velocity of the self-driving car.

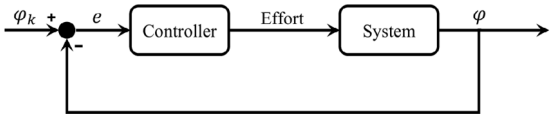


Fig. 12. Diagram of the feedback control.

4.6. Control

The **Controller** subsystem receives the trajectory generated by the Motion Planner subsystem, eventually modified by the Obstacle Avoider subsystem, and computes and sends Effort commands to the actuators of the steering wheel, throttle and brakes of the self-driving car in order to make the car execute the trajectory as best as the physical world allows. As mentioned in Section 4.4, there are two basic forms of defining a trajectory: (i) as a sequence of commands, i.e. the trajectory $T^c = \{c_1, c_2, \ldots, c_{|T^c|}\}$, where each command $c_k = (v_k, \varphi_k, \Delta t_k)$ and, for each c_k, v_k is the desired velocity at time k, φ_k is the desired steering angle at time k, and Δt_k is the duration of c_k; or (ii) as a sequence of states $T^s = \{s_1, s_2, \ldots, s_{|T^s|}\}$, where each state $s_k = (p_k, t_k)$ and, for each s_k, p_k is a pose, and t_k is the absolute time in which this pose is expected to be achieved. The implementations of the Controller subsystem that receives a T^c trajectory can be classified as direct hardware actuation control methods, while those that receive a T^s trajectory can be classified as path tracking methods.

4.6.1. Direct hardware actuation control methods

Direct hardware actuation control methods compute effort inputs to the steering, throttle, and brakes actuators of the car directly from the trajectory computed by the motion planner subsystem (Section 4.2), and try and mitigate inaccuracies caused primarily by the model of how the efforts influence the self-driving car velocity, v, and steering wheel angle, φ.

One of the most common direct hardware actuation control methods for self-driving cars is feedback control. It involves applying the efforts, observing v and φ, and adjusting future efforts to correct errors (the difference between the current T^c v_k and φ_k and the measured v and φ). Fig. 12 shows the diagram of feedback control, where the controller block may be composed by a variety of controllers and the system block is composed by the system plant to be controlled.

Funke et al. (2012) used a feedback control method to implement the controller subsystem of an Audi TTS. The hardware structures of the Audi TTS were designed to achieve hard real-time control at 200 Hz. This high-speed hardware enabled their controller subsystem to drive the car at up to 160 km/h. Ziegler, Bender, Schreiber et al. (2014) employed a feedback control method for the controller subsystem of Bertha. Bertha's controller subsystem was able to drive the car at up to 100 km/h through the 103 km long Bertha-Benz-Memorial-Route. Li et al. (2017) adopted the same control method in a Toyota Land Cruiser, which was able to drive the car at up to 25 km/h.

Another popular hardware actuation method for self-driving cars is the Proportional Integral Derivative (PID) control method. It involves specifying a desired hardware input and an error measure that accounts for how far the hardware output is from the desired hardware input (Aström & Murray, 2010). In possession of this information, a PID control method computes the hardware input efforts, which is K_p times the error measure (proportional to the error according to K_p), plus K_i times the integral of the error, plus K_d times the derivative of the error, where K_p, K_i and K_d are parameters of the PID control method. Similar to the feedback control technique, the PID control method can deal with the current error, besides the accumulated past errors over time (Guidolini et al., 2017). Zhao et al. (2012) used an adaptive PID method based on the generalized minimum variance technique to implement the controller subsystem of the self-driving car Intelligent Pionner. Intelligent Pionner's controller subsystem was able to drive the car at up to 60 km/h.

Levinson et al. (2011) employed a mixture of a Model Predictive Control (MPC) method and a feedforward PID control technique for the controller subsystem of the robotic car Junior (Montemerlo et al., 2008) (Stanford University's car that finished in second place in the 2007 DARPA Urban Challenge). The controller subsystem receives as input T^s and computes the efforts to actuate on the throttle, braking and steering. MPC methods use models of how self-driving cars respond to efforts over time to predict future values of v or φ. They use these predictions to try and decide which efforts should be sent to the car now to better reduce current and future errors in v or φ. Levinson et al. (2011) mixture of MPC and PID was able to drive Junior up to 56 km/h. Guidolini et al. (2017) proposed a Neural based Model Predictive Control (N-MPC) method to tackle delays in the steering wheel hardware of the self-driving car IARA. They used the MPC method to reduce the impact of steering hardware delays by anticipating efforts that would timely move the car according to the current trajectory. They modeled IARA's steering hardware using a neural network and employed the neural-based steering model in the N-MPC steering control method. The proposed solution was compared to the standard solution based on the PID control method. The PID controller subsystem worked well for speeds of up to 25 km/h. However, above this speed, delays of IARA's steering hardware were too high to allow proper operation. The N-MPC subsystem reduced the impact of IARA's steering hardware delays, which allowed its autonomous operation at speeds of up to 37 km/h – an increase of 48% in previous maximum stable speed.

4.6.2. Path tracking methods

Path tracking methods stabilize (i.e., reduce the error in) the execution of the motion plan computed by the motion planner subsystem (Section 4.2), in order to reduce inaccuracies caused mainly by car's motion model. They can be considered simplified trajectory planning techniques, although they do not deal with obstacles. The pure pursuit method (Paden et al., 2016) is broadly used in self-driving cars for path tracking, because of its simple implementation. It consists of finding a point in the path at some look-ahead distance from the current path and turning the front wheel so that a circular arc connects the center of the rear axle with the point in the path (Paden et al., 2016), as shown in Fig. 13. There is a set of variants proposed to improve the pure pursuit method. Samson (1995) propose to use the rear wheel position as the hardware output to stabilize the execution of the motion plan. Thrun et al. (2006) present the control approach of the self-driving car Stanley, which consists of taking the front wheel position as the regulated variable. Stanley's controller subsystem was able to drive the car at up to 60 km/h.

The Model Predictive Control (MPC) method is widely employed in self-driving cars. It consists of selecting control command inputs that

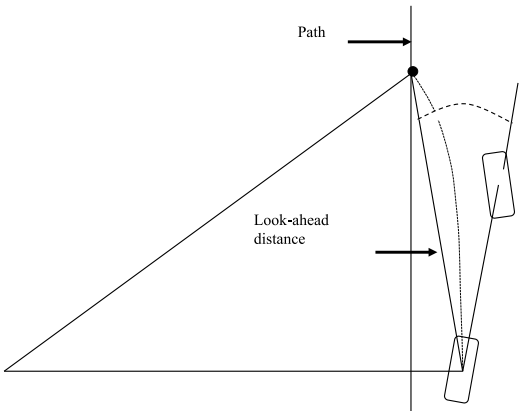


Fig. 13. Pure pursuit approach geometry.

Fig. 14. Intelligent and Autonomous Robotic Automobile (IARA) that was the first Brazilian self-driving car to travel autonomously 74 km on urban roads and highways. A video that shows IARA operating autonomously is available at https://youtu.be/iyKZV0lCysc.

will lead to desired hardware outputs, using the car's motion model to simulate and optimize outputs over a prediction horizon in the future (Guidolini et al., 2017). Koga et al. (2016) use the MPC method for path tracking of a small electrical car. They employ the standard bicycle model to predict car's motion. The electrical car's controller subsystem was able to drive the car at up to 20 km/h. Kritayakirana and Gerdes (2012) present the control approach of an Audi TTS, in which a dynamic motion model of the car (Brown et al., 2017; Laurense et al., 2017) is used to enable path tracking at the limits of handling. The car's dynamic motion model considers the deviation from the path plan due to tire slippage. Experimental results demonstrated that the Audi TTS could achieve a speed of up to 160 km/h.

5. Architecture of the UFES's car, IARA

In this section, to put everything in context, we present a detailed description of the architecture of a research self-driving car, the Intelligent Autonomous Robotic Automobile (IARA). IARA (Fig. 14) follows the typical architecture of self-driving cars. It was developed at the Laboratory of High Performance Computing (Laboratório de Computação de Alto Desempenho – LCAD) of the Federal University of Espírito Santo (Universidade Federal do Espírito Santo – UFES). IARA was the first Brazilian self-driving car to travel autonomously 74 km on urban roads and highways. The 74 km route from Vitória to Guarapari was traveled by IARA at the night of May 12, 2017.

5.1. Architecture of IARA's hardware

IARA is based on a 2011 Ford Escape Hybrid, which was adapted for autonomous driving. The actuation on the steering wheel is made by the factory-installed electric power steering system, which was modified so that an added electronic module can send signals equivalent to those sent by the original torque sensor to the electric power steering system. The electronic module also sends signals equivalent to the signals the throttle pedal sends to the car's electronics. The actuation on the brakes is made by an electric linear actuator attached to the brakes' pedal. The breaks' pedal can be used normally even in autonomous mode, though, thanks to the way the linear actuator is attached to it (the actuator only touches the pedal and can push it). The gear stick is originally attached to a series of electric switches that informs the car's electronics the current gear. A series of relays we have installed allows either connecting our autonomy system to the car's electronics or the original switches controlled by the gear stick. Similarly, the electric power steering wires and throttle pedal wires can be connected to the original car's wiring or to our autonomy system. At the press of a button, we can select the original manual factory operation or autonomous operation.

To power the computers and sensors that implement the autonomy system, we take advantage of the 330 V battery used by the electric powertrain of the Ford Escape Hybrid. We use a DC-DC converter to reduce it to 54 V; this DC-DC converter charges the battery of standard no-breaks that provides 120 V AC inside the car. These no-breaks can also be connected to the 120 V AC mains, which allow powering the computers and sensors of IARA while it is in the LCAD garage without requiring to turn on the Ford Escape Hybrid's engine.

IARA's computers and sensors comprise a workstation (Dell Precision R5500, with 2 Xeon X5690 six-core 3.4 GHz processors and one NVIDIA GeForce GTX-1030), networking gear, two LIDARs (a Velodyne HDL-32E and a SICK LD-MRS), three cameras (two Bumblebee XB3 and one ZED), an IMU (Xsens MTi), and a dual RTK GPS (based on the Trimble BD982 receiver).

5.2. Architecture of IARA's software

Following the typical system architecture of self-driving cars, IARA's software architecture is organized into perception and decision making systems, as shown in Fig. 1. Each subsystem of Fig. 1 is implemented in IARA as one or more software modules. Fig. 15 shows, as a block diagram, the software modules that implement the software architecture of IARA. IARA's software receives data from the sensors (in yellow in Fig. 15) that are made available through drivers modules (in red in Fig. 15) and sends them to the filter modules (green in Fig. 15). The filter modules receive as input data from sensors and/or other filter modules, and generate as output processed versions of this data. The filter modules implement, individually or in groups, the subsystems of Fig. 1.

The **GNSS** module transforms latitude and longitude data of the IARA Dual RTK GNSS into UTM coordinates (x, y, z) and synchronizes this information with the IARA orientation data (yaw - a Dual RTK GNSS has two antennas and centimeter accuracy, which makes it possible to obtain IARA's yaw with respect to the true north of Earth). The data in this module is used by the Localizer subsystem when the global localization and by the Mapper subsystem when generating the offline map. The **Stereo** module (de Paula Veronese et al., 2012) produces 3D depth maps (Point Cloud) from stereo images. It can be used by the 2D Mapper module as an alternative to LIDARs. The **Map Server** module (Mutz et al., 2016) provides the current Offline Map, given the current IARA's pose, extracted from the Offline Map database previously computed using the GraphSLAM algorithm. It is part of the Mapper subsystem of Fig. 1. The **Localizer** module (Veronese et al., 2015, 2016) localizes IARA on the current Offline Map from IARA (v, phi) and Point Clouds data from LIDAR Velodyne, and provides IARA

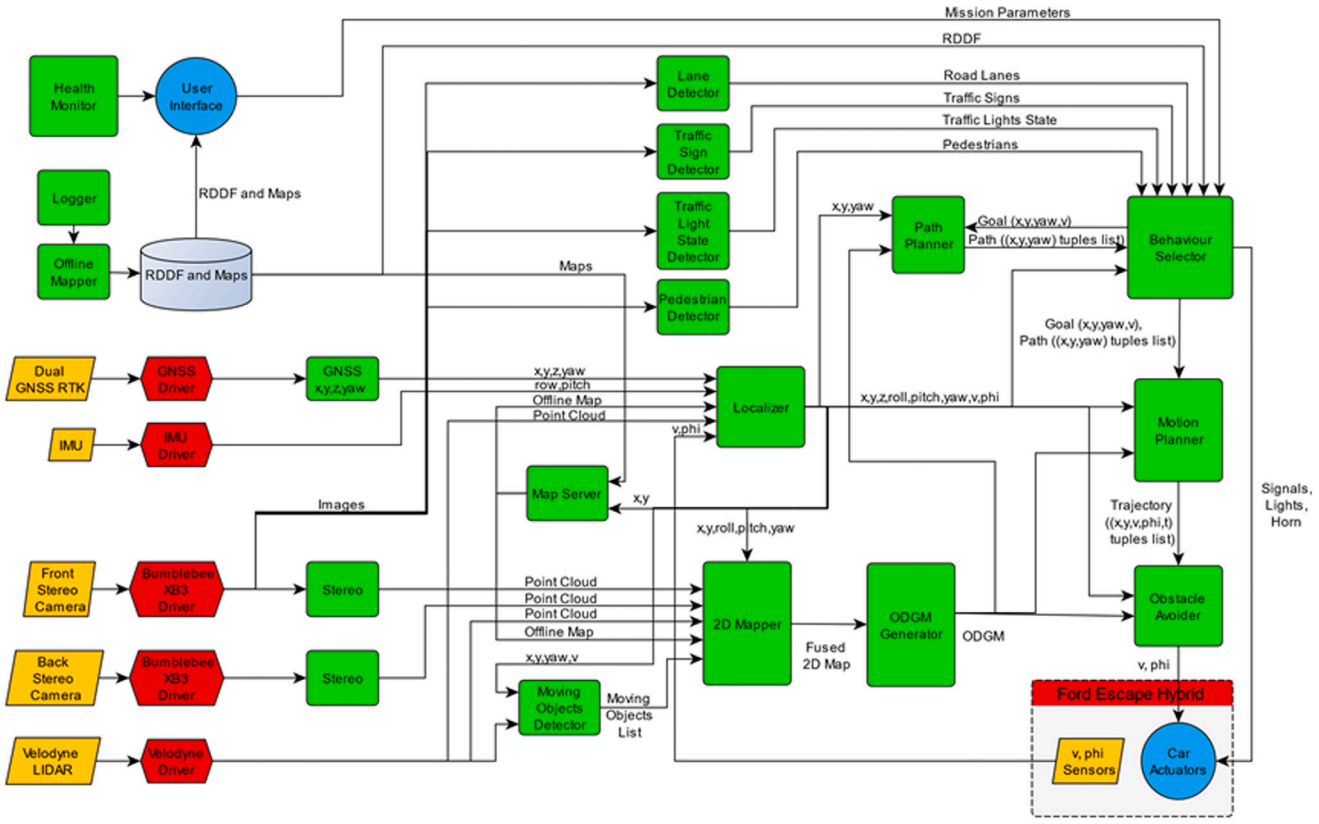


Fig. 15. Block diagram of the software modules that implement the software architecture of IARA.

status with 8 dimensions (x, y, z, roll, pitch, yaw, v, phi). Note that the pose provided by GNSS is only used when IARA is initialized (during the global localization process, when it is not done by imaging (Lyrio et al., 2015, 2014)) and the IMU data is only synchronized with the computed position. The Localizer module implements the Localizer subsystem. The **Moving Objects Detector** module (Amaral et al., 2015; Guidolini et al., 2018; Sarcinelli et al., 2019), using the Velodyne's Point Cloud and the pose and velocity of IARA, detects moving obstacles (x, y, yaw and geometry position) and its speed. It implements the MOT subsystem. The **2D Mapper** module creates an online map of the path and static and moving obstacles around IARA (Fused 2D Map) using IARA's status, various Point Clouds, the offline map and a list of Moving Objects List (see video at https://youtu.be/cyIfGupar-8). It is part of the Mapper subsystem. The **ODGM Generator** module generates ODGM from the Fused 2D Map. It is part of the Mapper subsystem.

The **Lane Detector** module (Berriel, de Aguiar et al., 2017) identifies the path markings on the lane (horizontal traffic signaling, or Road Lanes in Fig. 15) from the images captured by the front camera (see videos at https://youtu.be/NPU9tiyA8vw and https://youtu.be/R5wdPJ4ZI5M). It is part of the TSD subsystem. The **Traffic Sign Detector** module (De Souza et al., 2013; Torres et al., 2019) detects and recognizes (Berger et al., 2013) traffic signs along the path from images captured by the front camera (see video at https://youtu.be/SZ9w1XBWJqE). It is part of the TSD subsystem. The **Traffic Light State Detector** module (Possatti et al., 2019) detects traffic lights along the path and identifies its status from images captured by the front camera (see video at https://youtu.be/VjnCu3YXl2o). It is part of the TSD subsystem.

The **Path Planner** module, on demand of the Behavior Selector module, computes a Path ((x, y, yaw) tuple list) from the current IARA's position to a goal (Goal (x, y, yaw, v)). It implements the Path Planner subsystem. The **Behavior Selector** module, considering information provided by several modules, establishes a goal for IARA (Goal, i.e., x, y, yaw, v) and suggests a path to this goal (Path ((x, y, yaw) tuple list)). It implements the Behavior Selector subsystem. The **Motion Planner** module (Cardoso et al., 2017), from a goal (Goal (x, y, yaw, v)), a suggested path (Path ((x, y, yaw) y, z, roll, pitch, yaw, v, phi), constructs a trajectory (Trajectory ((x, y, v, phi, t) tuples list) from the current IARA's position to the target, but considering any obstacles present in the online map (https://youtu.be/tg8tkciHGzc) It implements the Motion Planner subsystem. The **Obstacle Avoider** module (Guidolini et al., 2016) evaluates continuously and at high speed the path generated by the Motion Planner module, changing, if necessary, their velocities, in order to avoid eminent collisions (https://youtu.be/6X_8q73g2kA). It implements the Obstacle Avoider subsystem. The **Ford Escape Hybrid** module (Guidolini et al., 2017) is the driver of the Ford Escape Hybrid car. From the outputs of the Obstacle Avoider and the Behavior Selector modules, it generates the signals that control the car. It implements the Controller subsystem. Finally, the **Health Monitor** module continuously checks for the proper functioning of all modules, automatically restarts modules that are not operating properly, and tells the user about their actions.

6. Self-driving cars under development in the industry

In this section, we list prominent self-driving car research platforms developed by academia and technology companies, and reported in the media. Several companies demonstrated interest in developing self-driving cars, and/or investing in technology to support and profit from them. Enterprises range from manufacturing cars and creating hardware for sensing and computing to developing software for assisted and autonomous driving, entertainment and in-car advertisement. We provide an overview of companies doing research and development in self-driving cars. The list is not presented in any specific order, since we aim at making it as impartial and complete as possible. The information was acquired by inspecting companies' websites and news published in other media.

Torc was one of the pioneers in developing cars with autonomous capabilities. The company was founded in 2005. In 2007, it joined Virginia Tech's team to participate in the 2007 DARPA Urban Challenge with their self-driving car Odin (Bacha et al., 2008), which reached third place in the competition. The technology used in the competition was improved since then and it was successfully applied in a variety of commercial ground vehicles, from large mining trucks to military vehicles.

Google's self-driving car project began in 2009 and was formerly led by Sebastian Thrun, who also led the Stanford University's team with their car Stanley (Thrun et al., 2006), winner of the 2005 DARPA Grand Challenge. In 2016, Google's self-driving car project became an independent company called Waymo. The company is a subsidiary of the holding Alphabet Inc., which is also Google's parent company. Waymo's self-driving car uses a sensor set composed of LIDARs to create a detailed map of the world around the car, RADARs to detect distant objects and their velocities, and high resolution cameras to acquire visual information, such as whether a traffic signal is red or green.

Baidu, one of the giant technology companies in China, is developing an open source self-driving car project with codename Apollo. The source code for the project is available in GitHub. It contains modules for perception (detection and tracking of moving obstacles, and detection and recognition of traffic lights), HD map and localization, planning, and control, among others. Several companies are partners of Baidu in the Apollo project, such as TomTom, Velodyne, Bosch, Intel, Daimler, Ford, Nvidia, and Microsoft. One of the Apollo project's goals is to create a centralized place for Original Equipment Manufacturers (OEMs), startups, suppliers, and research organizations to share and integrate their data and resources. Besides Baidu, Udacity, an educational organization founded by Sebastian Thrun and others, is also developing an open source self-driving car project, which is available for free in GitHub.

Uber is a ride-hailing service and, in 2015, they partnered with the Carnegie Mellon University to develop self-driving cars. A motivation for Uber's project is to replace associated drivers by autonomous software.

Lyft is a company that provides ridesharing and on-demand driving services. Like Uber, Lyft is doing research and development in self-driving cars. The company aims at developing cars with level 5 of autonomy.

Aptiv is one of Lyft's partners. Aptiv was created in a split of Delphi Automotive, a company owned by General Motors. Aptiv's objective is to build cars with level 4 and, posteriorly, level 5 of autonomy. Besides other products, the company sells short-range communication modules for vehicle-to-vehicle information exchange. Aptiv recently acquired two relevant self-driving car companies, Movimento and nuTonomy.

Didi is a Chinese transportation service that bought Uber's rights in China. Didi's self-driving car project was announced in 2017 and, in February of 2018, they did the first successful demonstration of their technology. In the same month, company's cars started being tested in USA and China. Within a year, Didi obtained the certificate for HD mapping in China. The company is now negotiating a partnership with Renault, Nissan, and Mitsubishi to build an electric and autonomous ride-sharing service.

Tesla was founded in 2003 and, in 2012, it started selling its first electric car, the Model S. In 2015, the company enabled the autopilot software for owners of the Model S. The software has been improved since then and its current version, the so called Tesla Autopilot, is now able to match speed with traffic conditions, keep within a lane, change lanes, transition from one freeway to another, exit the freeway when the destination is near, self-park when near a parking spot, and be summoned to and from the user's garage. Considering the limitation of basically operating only in highways (among others), Tesla Autopilot can be considered level 2. Tesla's current sensor set does not include LIDARs.

A Chinese company called LeEco is producing self-driving luxury sedans to compete with the Tesla Model S. The company is also backing up Faraday Future for the development of a concept car. LeEco is also partnering Aston Martin for the development of the RapidE electric car.

Besides developing hardware for general-purpose high-performance computing, NVDIA is also developing hardware and software for self-driving cars. Although their solutions rely mostly on artificial intelligence and deep learning, they are also capable of performing sensor fusion, localization in HD maps, and planning.

Aurora is a new company founded by experienced engineers that worked in Google's self-driving car project, Tesla, and Uber. The company plans to work with automakers and suppliers to develop full-stack solutions for cars with level 4 and, eventually, level 5 of autonomy. Aurora has independent partnerships with Volkswagen group (that owns Volkswagen Passenger Cars, Audi, Bentley, Skoda, and Porsche) and Hyunday.

Zenuity is a joint venture created by Volvo Cars and Autoliv. Ericsson will aid in the development of Zenuity's Connected Cloud that will use Ericsson's IoT Accelerator. TomTom also partnered with Zenuity and provided its HD mapping technology. TomTom's HD maps will be used for localization, perception and path planning in Zenuity's software stack.

Daimler and Bosch are joining forces to advance the development of cars with level 4 and 5 of autonomy by the beginning of the next decade. The companies already have an automated valet parking in Stuttgart and they also have tested the so called Highway Pilot in trucks in USA and Germany. Besides partnering with Bosch, Daimler also merged its car sharing business, Car2Go, with BMW's ReachNow, from the same business segment, in an effort to stave off competition from other technology companies, such as Waymo and Uber. The new company will include business in car sharing, ride-hailing, valet parking, and electric vehicle charging.

Argo AI founders led self-driving car teams at Google and Uber. The company received an investment of U$1 billion from Ford with the goal of developing a new software platform for Ford's fully autonomous car (level 4) coming in 2021. They have partnerships with professors from Carnegie Mellon University and Georgia Tech.

Renesas Autonomy develops several solutions for Automated Driving Assistant Systems (ADAS) and automated driving. The company has a partnership with the University of Waterloo.

Honda revealed in 2017 plans for introducing cars with level 4 of autonomy by 2025. The company intends to have cars with level 3 of autonomy by 2020 and it is negotiating a partnership with Waymo.

Visteon is a technology company that manufactures cockpit electronic products and connectivity solutions for several vehicle manufacturers. In 2018, Visteon introduced its autonomous driving platform capable of level 3 of autonomy and, potentially, higher levels. The company does not aim at producing cars and sensors, but at producing integrated solutions and software.

AImotive is using low cost components to develop a self-driving car. Its solution relies strongly on computer vision, but it uses additional sensors. The company is already able to perform valet parking and navigate in highways. AImotive also developed a photorealistic simulator for data collection and preliminary system testing, and chips for artificial intelligence-based, latency-critic, and camera-centric systems.

As AImotive, AutoX is avoiding to use LIDARs in its solution. However, the company is going further than Almotive and trying to develop a level 5 car without using RADARs, ultrasonics, and differential GPS's. AutoX's approach is to create a full-stack software solution based on artificial intelligence.

Mobileye is also seeking to develop a self-driving car without using LIDARs, but relying mostly in a single-lensed camera (mono-camera). Mobileye is one of the leading suppliers of software for Advanced Driver Assist Systems (ADAS), with more than 25 partners among automakers. Beyond ADAS, Mobileye is also developing technology to

support other key components for autonomous driving, such as perception (detection of free space, driving paths, moving objects, traffic lights, and traffic signs, among others), mapping, and control. The company partnered with BMW and Intel to develop production-ready fully autonomous cars, with production launch planned for 2021.

Ambarella also does not use LIDAR, but only RADARs and stereo cameras. The company joined the autonomous driving race in 2015 by acquiring VisLAB. Different from other companies, Ambarella does not aim at becoming a tier-one supplier or selling fully autonomous driving systems. Instead, they plan to sell chips and software to automakers, suppliers, and software developers. Ambarella's current research and development guidelines include detection of vehicles, obstacles, pedestrians, and lanes; traffic sign recognition; terrain mapping; and issues related to technology commercialization, such as system calibration, illumination, noise, temperature, and power consumption.

Pony.ai was founded in December of 2016 and, in July of 2017, it completed its first fully autonomous driving demonstration. The company signed a strategic agreement with one of the biggest Chinese car makers, the Guangzhou Auto Group (GAC).

Navya and Transdev are French companies that develop self-driving buses. Navya has several of their buses being tested in Europe, Asia, and Australia. Their sensor set consists of two multi-layer 360° LIDARs, six 180° mono-layer LIDARs, front and rear cameras, odometer (wheels encoder + IMU), and a GNSS RTK. Transdev is also already demonstrating their self-driving buses for the public.

JD is a Chinese e-commerce company interested in building autonomous delivery vehicles. JD's project, started in 2016, is being developed together with Idriverplus, a Chinese self-driving car startup.

In March, 2018, Toyota announced an investment of U $2.8 billion in the creation of a new company called the Toyota Research Institute-Advanced Development (TRI-AD) with the goal of developing an electric and self-driving car until 2020. Besides Toyota, other car manufacturers, such as Ford, Volvo, and Mercedes-Benz, have also recently presented their plans for self-driving cars. Ford defined 2021 as a deadline for presenting a fully autonomous car ready for commercial operation.

7. Conclusion

In this paper, we surveyed the literature on self-driving cars focusing on research that has been tested in the real world. Analyzing this body of literature, we were able to present a detailed view of the typical architecture of the autonomy system of self-driving cars, clearly describing each one of its main components. This architecture is organized into two main parts: the perception system and the decision-making system. The perception system is generally divided into many subsystems responsible for tasks such as self-driving car location, mapping of static obstacles, road mapping, detection and tracking of moving obstacles, detection and recognition of traffic signs, among others, while the decision-making system is generally divided into subsystems responsible for tasks such as route planning, path planning, behavior selection, motion planning, control and obstacle avoidance, although this partitioning of the decision-making system is somewhat blurred and there are several variations in literature. The main contribution of this paper is the detailed description of the architecture of self-driving cars and the discussion of the large body of research on each of its components.

Another contribution of this paper is the description of the IARA's autonomy system. IARA is an advanced research self-driving car we have been developing since 2009, and its description in the paper (Section 5) allows putting in context the many aspects of what is discussed in the literature review. It is worth mentioning that the IARA's autonomy system was used recently, almost without modifications, to make a passenger jet (an Embraer Legacy 500) capable of taxiing autonomously (https://www.youtube.com/watch?v=vC5UiYKVsic), which suggests that the architecture of the autonomy system for self-driving cars we present is general, correct and useful. Porting IARA's autonomy system to the Legacy 500 took a small team of researchers (8 people) just 10 months. This result implies that the same general architecture can be used in many different contexts.

Autonomous cars are perhaps the most advanced intelligent systems developed so far in the world. They need to monitor the environment around them, build an internal representation of it and use this internal representation to interact intelligently with the environment and the people in it. Since the DARPA challenges of 2004, 2005 and 2007, a large body of research has contributed to the current state of self-driving cars' technology. However, much still has to be done to achieve the industry and academy goal of making SAE level 4 or, hopefully, level 5, self-driving cars available to the public at large.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil, Grants 311654/2019-3 and 311504/2017-5; Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Brazil , Finance Code 001; Fundação de Amparo à Pesquisa do Espírito Santo (FAPES), Brazil, Grant 84412844/2018; Vale company, Brazil, with FAPES, Brazil, Grant 75537958/16; and Embraer company, Brazil, Grant GDT0017-18.

References

Abraham, I., Delling, D., Goldberg, A. V., & Werneck, R. F. (2012). Hierarchical hub labelings for shortest paths. In *European symposium on algorithms* (pp. 24–35). Springer.

Aeberhard, M., Rauch, S., Bahram, M., Tanzmeister, G., Thomas, J., Pilat, Y., Homm, F., Huber, W., & Kaempchen, N. (2015). Experience, results and lessons learned from automated driving on germany's highways. *IEEE Intelligent Transportation Systems Magazine*, *7*(1), 42–57.

Ahmad, T., Ilstrup, D., Emami, E., & Bebis, G. (2017). Symbolic road marking recognition using convolutional neural networks. In *2017 IEEE intelligent vehicles symposium (IV)* (pp. 1428–1433). IEEE.

Alia, C., Gilles, T., Reine, T., & Ali, C. (2015). Local trajectory planning and tracking of autonomous vehicles, using clothoid tentacles method. In *2015 IEEE intelligent vehicles symposium (IV)* (pp. 674–679). IEEE.

Amaral, E., Badue, C., Oliveira-Santos, T., & De Souza, A. F. (2015). Detecção e Rastreamento de Veículos em Movimento para Automóveis Robóticos Autônomos. In *XII simpósio brasileiro de automação inteligente (SBAI)* (pp. 801–806).

Arnay, R., Morales, N., Morell, A., Hernandez-Aceituno, J., Perea, D., Toledo, J. T., Hamilton, A., Sanchez-Medina, J. J., & Acosta, L. (2016). Safe and reliable path planning for the autonomous vehicle verdino. *IEEE Intelligent Transportation Systems Magazine*, *8*(2), 22–32.

Arz, J., Luxen, D., & Sanders, P. (2013). Transit node routing reconsidered. In *International symposium on experimental algorithms* (pp. 55–66). Springer.

Aström, K. J., & Murray, R. M. (2010). *Feedback systems: an introduction for scientists and engineers*. Princeton university press.

Awad, E., Dsouza, S., Kim, R., Schulz, J., Henrich, J., Shariff, A., Bonnefon, J.-F., & Rahwan, I. (2018). The moral machine experiment. *Nature*, *563*(7729), 59.

Azim, A., & Aycard, O. (2014). Layer-based supervised classification of moving objects in outdoor dynamic environment using 3d laser scanner. In *2014 IEEE intelligent vehicles symposium* (pp. 1408–1414). IEEE.

Bacha, A., Bauman, C., Faruque, R., Fleming, M., Terwelp, C., Reinholtz, C., Hong, D., Wicks, A., Alberi, T., & Anderson, D. (2008). Odin: Team victortango's entry in the darpa urban challenge. *Journal of Field Robotics*, *25*(8), 467–492.

Bailo, O., Lee, S., Rameau, F., Yoon, J. S., & Kweon, I. S. (2017). Robust road marking detection and recognition using density-based grouping and machine learning techniques. In *2017 IEEE winter conference on applications of computer vision (WACV)* (pp. 760–768). IEEE.

Barnes, N., Zelinsky, A., & Fletcher, L. S. (2008). Real-time speed sign detection using the radial symmetry detector. *IEEE Transactions on Intelligent Transportation Systems*, *9*(2), 322–332.

Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., & Werneck, R. F. (2015). Route planning in transportation networks. arXiv preprint arXiv:1504.05140.

Bastani, F., He, S., Abbar, S., Alizadeh, M., Balakrishnan, H., Chawla, S., Madden, S., & DeWitt, D. (2018). Roadtracer: Automatic extraction of road networks from aerial images. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4720–4728).

Bauer, R., & Delling, D. (2008). Sharc: Fast and robust unidirectional routing. In *2008 proceedings of the tenth workshop on algorithm engineering and experiments (ALENEX)* (pp. 13–26). SIAM.

Bauer, R., Delling, D., Sanders, P., Schieferdecker, D., Schultes, D., & Wagner, D. (2010). Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm. *ACM Journal of Experimental Algorithmics*, *15*(2.3).

Behrendt, K., Novak, L., & Botros, R. (2017). A deep learning approach to traffic lights: Detection, tracking, and classification. In *2017 IEEE international conference on robotics and automation (ICRA)* (pp. 1370–1377). IEEE.

Bender, P., Ziegler, J., & Stiller, C. (2014). Lanelets: Efficient map representation for autonomous driving. In *2014 IEEE intelligent vehicles symposium proceedings* (pp. 420–425). IEEE.

Berger, M., Forechi, A., De Souza, A. F., Neto, J. D. O., Veronese, L., Neves, V., de Aguiar, E., & Badue, C. (2013). Traffic sign recognition with wisard and vg-ram weightless neural networks. *Journal of Network and Innovative Computing*, *1*(1), 87–98.

Bernini, N., Bertozzi, M., Castangia, L., Patander, M., & Sabbatelli, M. (2014). Real-time obstacle detection using stereo vision for autonomous ground vehicles: A survey. In *17th international IEEE conference on intelligent transportation systems (ITSC)* (pp. 873–878). IEEE.

Berriel, R. F., de Aguiar, E., De Souza, A. F., & Oliveira-Santos, T. (2017). Ego-lane analysis system (elas): Dataset and algorithms. *Image and Vision Computing*, *68*, 64–75.

Berriel, R. F., de Aguiar, E., de Souza Filho, V. V., & Oliveira-Santos, T. (2015). A particle filter-based lane marker tracking approach using a cubic spline model. In *2015 28th SIBGRAPI conference on graphics, patterns and images* (pp. 149–156). IEEE.

Berriel, R. F., Rossi, F. S., de Souza, A. F., & Oliveira-Santos, T. (2017). Automatic large-scale data acquisition via crowdsourcing for crosswalk classification: A deep learning approach. *Computers & Graphics*, *68*, 32–42.

Brechtel, S., Gindele, T., & Dillmann, R. (2011). Probabilistic mdp-behavior planning for cars. In *2011 IEEE 14th International Conference on Intelligent Transportation Systems (ITSC)* (pp. 1537–1542).

Brechtel, S., Gindele, T., & Dillmann, R. (2014). Probabilistic decision-making under uncertainty for autonomous driving using continuous pomdps. In *17th international IEEE conference on intelligent transportation systems (ITSC)* (pp. 392–399). IEEE.

Broggi, A., Bertozzi, M., & Fascioli, A. (1999). Argo and the millemiglia in automatico tour. *IEEE Intelligent Systems and their Applications*, *14*(1), 55–64.

Broggi, A., Cerri, P., Debattisti, S., Laghi, M. C., Medici, P., Molinari, D., Panciroli, M., & Prioletti, A. (2015). Proud—public road urban driverless-car test. *IEEE Transactions on Intelligent Transportation Systems*, *16*(6), 3508–3519.

Broggi, A., Cerri, P., Felisa, M., Laghi, M. C., Mazzei, L., & Porta, P. P. (2012). The vislab intercontinental autonomous challenge: an extensive test for a platoon of intelligent vehicles. *International Journal of Vehicle Autonomous Systems*, *10*(3), 147–164.

Brown, M., Funke, J., Erlien, S., & Gerdes, J. C. (2017). Safe driving envelopes for path tracking in autonomous vehicles. *Control Engineering Practice*, *61*, 307–316.

Brubaker, M. A., Geiger, A., & Urtasun, R. (2015). Map-based probabilistic visual self-localization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *38*(4), 652–665.

Buehler, M., Iagnemma, K., & Singh, S. (2007). *The 2005 DARPA grand challenge: the great robot race, Vol. 36*. Springer.

Buehler, M., Iagnemma, K., & Singh, S. (2009). *The DARPA urban challenge: autonomous vehicles in city traffic, Vol. 56*. springer.

Cao, H., Song, X., Huang, Z., & Pan, L. (2016). Simulation research on emergency path planning of an active collision avoidance system combined with longitudinal control for an autonomous vehicle. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of automobile engineering*, *230*(12), 1624–1653.

Cardoso, V., Oliveira, J., Teixeira, T., Badue, C., Mutz, F., Oliveira-Santos, T., Veronese, L., & De Souza, A. F. (2017). A model-predictive motion planner for the iara autonomous car. In *2017 IEEE international conference on robotics and automation (ICRA)* (pp. 225–230). IEEE.

Carneiro, R. V., Nascimento, R. C., Guidolini, R., Cardoso, V. B., Oliveira-Santos, T., Badue, C., & De Souza, A. F. (2018). Mapping road lanes using laser remission and deep neural networks. In *2018 international joint conference on neural networks (IJCNN)* (pp. 1–8). IEEE.

Cerri, P., Soprani, G., Zani, P., Choi, J., Lee, J., Kim, D., Yi, K., & Broggi, A. (2011). Computer vision at the hyundai autonomous challenge. In *2011 14th international IEEE conference on intelligent transportation systems (ITSC)* (pp. 777–783). IEEE.

Chen, L., Fan, L., Xie, G., Huang, K., & Nüchter, A. (2017). Moving-object detection from consecutive stereo pairs using slanted plane smoothing. *IEEE Transactions on Intelligent Transportation Systems*, *18*(11), 3093–3102.

Chen, J., & Shen, S. (2017). Improving octree-based occupancy maps using environment sparsity with application to aerial robot navigation. In *2017 IEEE international conference on robotics and automation (ICRA)* (pp. 3656–3663). IEEE.

Cho, H., Seo, Y.-W., Kumar, B. V., & Rajkumar, R. R. (2014). A multi-sensor fusion system for moving object detection and tracking in urban driving environments. In *2014 IEEE international conference on robotics and automation (ICRA)* (pp. 1836–1843). IEEE.

Chu, K., Kim, J., Jo, K., & Sunwoo, M. (2015). Real-time path planning of autonomous vehicles for unstructured road navigation. *International Journal of Automotive Technology*, *16*(4), 653–668.

Chu, K., Lee, M., & Sunwoo, M. (2012). Local path planning for off-road autonomous driving with avoidance of static obstacles. *IEEE Transactions on Intelligent Transportation Systems*, *13*(4), 1599–1616.

Cohen, E., Halperin, E., Kaplan, H., & Zwick, U. (2003). Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, *32*(5), 1338–1355.

Conitzer, V., Sinnott-Armstrong, W., Borg, J. S., Deng, Y., & Kramer, M. (2017). Moral decision making frameworks for artificial intelligence. In *Thirty-first aaai conference on artificial intelligence*.

Cummins, M., & Newman, P. (2008). Fab-map: Probabilistic localization and mapping in the space of appearance. *International Journal of Robotics Research*, *27*(6), 647–665.

Darms, M. S., Rybski, P. E., Baker, C., & Urmson, C. (2009). Obstacle detection and tracking for the urban challenge. *IEEE Transactions on Intelligent Transportation Systems*, *10*(3), 475–485.

DARPA (2005). *DARPA grand challenge 2005 route data definition file: Technical report*, Arlington, VA, USA: Defense Advanced Research Projects Agency.

DARPA (2007). *Urban challenge: Route network definition file (RNDF) and mission data file (MDF) formats: Technical report*, Arlington, VA, USA: Defense Advanced Research Projects Agency.

De Lima, D. A., & Pereira, G. A. (2010). Um sistema de visao estéreo para navegaçao de um carro autônomo em ambientes com obstáculos. In *XVIII congresso brasileiro de automática* (pp. 224–231).

De Lima, D. A., & Pereira, G. A. S. (2013). Navigation of an autonomous car using vector fields and the dynamic window approach. *Journal of Control, Automation and Electrical Systems*, *24*(1–2), 106–116.

De Souza, A. F., Fontana, C., Mutz, F., de Oliveira, T. A., Berger, M., Forechi, A., de Oliveira Neto, J., de Aguiar, E., & Badue, C. (2013). Traffic sign detection with vg-ram weightless neural networks. In *The 2013 international joint conference on neural networks (IJCNN)* (pp. 1–9). IEEE.

Delling, D., Goldberg, A. V., Nowatzyk, A., & Werneck, R. F. (2013). Phast: Hardware-accelerated shortest path trees. *Journal of Parallel and Distributed Computing*, *73*(7), 940–952.

Delling, D., Goldberg, A. V., Pajor, T., & Werneck, R. F. (2015). Customizable route planning in road networks. *Transportation Science*, *51*(2), 566–591.

Delling, D., Goldberg, A. V., & Werneck, R. F. (2013). Hub label compression. In *International symposium on experimental algorithms* (pp. 18–29). Springer.

Delling, D., Holzer, M., Müller, K., Schulz, F., & Wagner, D. (2009). High-performance multi-level routing. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, *74*, 73–92.

Dias, J. E. A., Pereira, G. A. S., & Palhares, R. M. (2014). Longitudinal model identification and velocity control of an autonomous car. *IEEE Transactions on Intelligent Transportation Systems*, *16*(2), 776–786.

Diaz-Cabrera, M., Cerri, P., & Medici, P. (2015). Robust real-time traffic light detection and distance estimation using a single camera. *Expert Systems with Applications*, *42*(8), 3911–3923.

Diaz-Cabrera, M., Cerri, P., & Sanchez-Medina, J. (2012). Suspended traffic lights detection and distance estimation using color features. In *2012 15th international IEEE conference on intelligent transportation systems* (pp. 1315–1320). IEEE.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*(1), 269–271.

Doherty, K., Wang, J., & Englot, B. (2016). Probabilistic map fusion for fast, incremental occupancy mapping with 3d hilbert maps. In *2016 IEEE international conference on robotics and automation (ICRA)* (pp. 1011–1018). IEEE.

Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. *International Journal of Robotics Research*, *29*(5), 485–501.

Droeschel, D., Schwarz, M., & Behnke, S. (2017). Continuous mapping and localization for autonomous navigation in rough terrain using a 3d laser scanner. *Robotics and Autonomous Systems*, *88*, 104–115.

Du, M., Mei, T., Liang, H., Chen, J., Huang, R., & Zhao, P. (2016). Drivers' visual behavior-guided rrt motion planner for autonomous on-road driving. *Sensors*, *16*(1), 102.

Englund, C., Chen, L., Ploeg, J., Semsar-Kazerooni, E., Voronov, A., Bengtsson, H. H., & Didoff, J. (2016). The grand cooperative driving challenge 2016: boosting the introduction of cooperative automated vehicles. *IEEE Wireless Communications*, *23*(4), 146–152.

Ess, A., Schindler, K., Leibe, B., & Van Gool, L. (2010). Object detection and tracking for autonomous navigation in dynamic environments. *International Journal of Robotics Research*, *29*(14), 1707–1725.

Fassbender, D., Heinrich, B. C., & Wuensche, H.-J. (2016). Motion planning for autonomous vehicles in highly constrained urban environments. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 4708–4713). IEEE.

Ferguson, D., Howard, T. M., & Likhachev, M. (2008). Motion planning in urban environments. *Journal of Field Robotics*, *25*(11–12), 939–960.

Fernandes, L. C., Souza, J. R., Pessin, G., Shinzato, P. Y., Sales, D., Mendes, C., Prado, M., Klaser, R., Magalhaes, A. C., & Hata, A. (2014). Carina intelligent robotic car: architectural design and applications. *Journal of Systems Architecture*, *60*(4), 372–392.

Forechi, A., Oliveira-Santos, T., Badue, C., & De Souza, A. F. (2018). Visual global localization with a hybrid wnn-cnn approach. In *2018 international joint conference on neural networks (IJCNN)* (pp. 1–9). IEEE.

Foucher, P., Sebsadji, Y., Tarel, J.-P., Charbonnier, P., & Nicolle, P. (2011). Detection and recognition of urban road markings using images. In *2011 14th international IEEE conference on intelligent transportation systems (ITSC)* (pp. 1747–1752). IEEE.

Funke, J., Theodosis, P., Hindiyeh, R., Stanek, G., Kritatakirana, K., Gerdes, C., Langer, D., Hernandez, M., Müller-Bessler, B., & Huhnke, B. (2012). Up to the limits: Autonomous audi tts. In *2012 IEEE intelligent vehicles symposium* (pp. 541–547). IEEE.

Galceran, E., Cunningham, A. G., Eustice, R. M., & Olson, E. (2017). Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction: Theory and experiment. *Autonomous Robots*, *41*(6), 1367–1382.

Gao, X. W., Podladchikova, L., Shaposhnikov, D., Hong, K., & Shevtsova, N. (2006). Recognition of traffic signs based on their colour and shape features extracted using human vision models. *Journal of Visual Communication and Image Representation*, *17*(4), 675–685.

Garcia-Fidalgo, E., & Ortiz, A. (2015). Vision-based topological mapping and localization methods: A survey. *Robotics and Autonomous Systems*, *64*, 1–20.

Ge, Z., Wang, P., Wang, J., & Chen, Z. (2017). A 2.5 d grids based moving detection methodology for autonomous vehicle. In *2017 2nd international conference on robotics and automation engineering (ICRAE)* (pp. 403–407). IEEE.

Geisberger, R., Sanders, P., Schultes, D., & Vetter, C. (2012). Exact routing in large road networks using contraction hierarchies. *Transportation Science*, *46*(3), 388–404.

Girao, P., Asvadi, A., Peixoto, P., & Nunes, U. (2016). 3d object tracking in driving environment: a short review and a benchmark dataset. In *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)* (pp. 7–12). IEEE.

Goldberg, A. V., & Harrelson, C. (2005). Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on discrete algorithms* (pp. 156–165). Society for Industrial and Applied Mathematics.

Goldberg, A. V., Kaplan, H., & Werneck, R. F. (2006). Reach for a*: Shortest path algorithms with preprocessing.. In *The shortest path problem* (pp. 93–140).

Gomez, A. E., Alencar, F. A., Prado, P. V., Osorio, F. S., & Wolf, D. F. (2014). Traffic lights detection and state estimation using hidden markov models. In *2014 IEEE intelligent vehicles symposium proceedings* (pp. 750–755). IEEE.

Gong, J., Jiang, Y., Xiong, G., Guan, C., Tao, G., & Chen, H. (2010). The recognition and tracking of traffic lights based on color segmentation and camshift for intelligent vehicles. In *2010 IEEE intelligent vehicles symposium* (pp. 431–435). Ieee.

González, D., Pérez, J., Milanés, V., & Nashashibi, F. (2015). A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, *17*(4), 1135–1145.

Greene, J., Rossi, F., Tasioulas, J., Venable, K. B., & Williams, B. (2016). Embedding ethical principles in collective decision support systems. In *Thirtieth AAAI conference on artificial intelligence*.

Greenhalgh, J., & Mirmehdi, M. (2015). Detection and recognition of painted road surface markings. In *Proceedings of the international conference on pattern recognition applications and methods-volume 1* (pp. 130–138). Lda: SCITEPRESS-Science and Technology Publications.

Gregor, R., Lutzeler, M., Pellkofer, M., Siedersberger, K.-H., & Dickmanns, E. D. (2002). Ems-vision: A perceptual system for autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, *3*(1), 48–59.

Gu, T., Atwood, J., Dong, C., Dolan, J. M., & Lee, J.-W. (2015). Tunable and stable real-time trajectory planning for urban autonomous driving. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 250–256). IEEE.

Gu, T., Dolan, J. M., & Lee, J.-W. (2016). Automated tactical maneuver discovery, reasoning and trajectory planning for autonomous driving. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 5474–5480). IEEE.

Gudigar, A., Chokkadi, S., & Raghavendra, U. (2016). A review on automatic detection and recognition of traffic sign. *Multimedia Tools and Applications*, *75*(1), 333–364.

Guidolini, R., Badue, C., Berger, M., de Paula Veronese, L., & De Souza, A. F. (2016). A simple yet effective obstacle avoider for the iara autonomous car. In *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)* (pp. 1914–1919). IEEE.

Guidolini, R., De Souza, A. F., Mutz, F., & Badue, C. (2017). Neural-based model predictive control for tackling steering delays of autonomous cars. In *2017 international joint conference on neural networks (IJCNN)* (pp. 4324–4331). IEEE.

Guidolini, R., Scart, L. G., Jesus, L. F., Cardoso, V. B., Badue, C., & Oliveira-Santos, T. (2018). Handling pedestrians in crosswalks using deep neural networks in the iara autonomous car. In *2018 international joint conference on neural networks (IJCNN)* (pp. 1–8). IEEE.

Gurghian, A., Koduri, T., Bailur, S. V., Carey, K. J., & Murali, V. N. (2016). Deeplanes: End-to-end lane position estimation using deep neural networksa. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* (pp. 38–45).

Gutman, R. J. (2004). Reach-based routing: a new approach to shortest path algorithms optimized for road networks. *ALENEX/ANALC*, *4*, 100–111.

Haltakov, V., Mayr, J., Unger, C., & Ilic, S. (2015). Semantic segmentation based traffic light detection at day and at night. In *German conference on pattern recognition* (pp. 446–457). Springer.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, *8*(3), 231–274.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Hata, A. Y., Ramos, F. T., & Wolf, D. F. (2017). Monte carlo localization on gaussian process occupancy maps for urban environments. *IEEE Transactions on Intelligent Transportation Systems*, *19*(9), 2893–2902.

Hata, A. Y., & Wolf, D. F. (2015). Feature detection for vehicle localization in urban environments using a multilayer lidar. *IEEE Transactions on Intelligent Transportation Systems*, *17*(2), 420–429.

Hata, A. Y., Wolf, D. F., & Ramos, F. T. (2016). Particle filter localization on continuous occupancy maps. In *International symposium on experimental robotics* (pp. 742–751). Springer.

He, X., Liu, Y., Lv, C., Ji, X., & Liu, Y. (2019). Emergency steering control of autonomous vehicle for collision avoidance and stabilisation. *Vehicle System Dynamics*, *57*(8), 1163–1187.

He, M., Takeuchi, E., Ninomiya, Y., & Kato, S. (2016). Precise and efficient model-based vehicle tracking method using rao-blackwellized and scaling series particle filters. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 117–124). IEEE.

Held, D., Thrun, S., & Savarese, S. (2016). Learning to track at 100 fps with deep regression networks. In *European conference on computer vision* (pp. 749–765). Springer.

Hess, W., Kohler, D., Rapp, H., & Andor, D. (2016). Real-time loop closure in 2d lidar slam. In *2016 IEEE international conference on robotics and automation (ICRA)* (pp. 1271–1278). IEEE.

Hilger, M., Köhler, E., Möhring, R. H., & Schilling, H. (2009). Fast point-to-point shortest path computations with arc-flags. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, *74*, 41–72.

Hillel, A. B., Lerner, R., Levi, D., & Raz, G. (2014). Recent progress in road and lane detection: a survey. *Machine Vision and Applications*, *25*(3), 727–745.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, *34*(3), 189–206.

Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M., & Igel, C. (2013). Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *The 2013 international joint conference on neural networks (IJCNN)* (pp. 1–8). IEEE.

Howard, T. M., & Kelly, A. (2007). Optimal rough terrain trajectory generation for wheeled mobile robots. *International Journal of Robotics Research*, *26*(2), 141–166.

Hu, X., Chen, L., Tang, B., Cao, D., & He, H. (2018). Dynamic path planning for autonomous driving on various roads with avoidance of static and moving obstacles. *Mechanical Systems and Signal Processing*, *100*, 482–500.

Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., Andriluka, M., Rajpurkar, P., Migimatsu, T., & Cheng-Yue, R. (2015). An empirical evaluation of deep learning on highway driving. arXiv preprint arXiv:1504.01716.

Hwang, S., Kim, N., Choi, Y., Lee, S., & Kweon, I. S. (2016). Fast multiple objects detection and tracking fusing color camera and 3d lidar for intelligent vehicles. In *2016 13th international conference on ubiquitous robots and ambient intelligence (URAI)* (pp. 234–239). IEEE.

Hyeon, D., Lee, S., Jung, S., Kim, S.-W., & Seo, S.-W. (2016). Robust road marking detection using convex grouping method in around-view monitoring system. In *2016 IEEE intelligent vehicles symposium (IV)* (pp. 1004–1009). IEEE.

Ivanchenko, V., Coughlan, J., & Shen, H. (2008). Detecting and locating crosswalks using a camera phone. In *2008 IEEE computer society conference on computer vision and pattern recognition workshops* (pp. 1–8). IEEE.

Jang, C., Kim, C., Kim, D., Lee, M., & Sunwoo, M. (2014). Multiple exposure images based traffic light recognition. In *2014 IEEE intelligent vehicles symposium proceedings* (pp. 1313–1318). IEEE.

Jensen, M. B., Nasrollahi, K., & Moeslund, T. B. (2017). Evaluating state-of-the-art object detector on challenging traffic light data. In *2017 IEEE conference on computer vision and pattern recognition workshops (CVPRW)* (pp. 882–888). IEEE.

Jensen, M. B., Philipsen, M. P., Møgelmose, A., Moeslund, T. B., & Trivedi, M. M. (2016). Vision for looking at traffic lights: Issues, survey, and perspectives. *IEEE Transactions on Intelligent Transportation Systems*, *17*(7), 1800–1815.

Jo, K., Jo, Y., Suhr, J. K., Jung, H. G., & Sunwoo, M. (2015). Precise localization of an autonomous car based on probabilistic noise models of road surface marker features using multiple cameras. *IEEE Transactions on Intelligent Transportation Systems*, *16*(6), 3377–3392.

Jo, K., Kim, J., Kim, D., Jang, C., & Sunwoo, M. (2015). Development of autonomous car-part ii: a case study on the implementation of an autonomous driving system based on distributed architecture. *IEEE Transactions on Industrial Electronics*, *62*(8), 5119–5132.

Jung, C. R., & Kelber, C. R. (2005). Lane following and lane departure using a linear-parabolic model. *Image and Vision Computing*, *23*(13), 1192–1202.

Kala, R., & Warwick, K. (2013). Multi-level planning for semi-autonomous vehicles in traffic scenarios based on separation maximization. *Journal of Intelligent and Robotic Systems*, *72*(3–4), 559–590.

Kim, S., & Kim, J. (2013). Continuous occupancy maps using overlapping local gaussian processes. In *2013 IEEE/RSJ international conference on intelligent robots and systems* (pp. 4709–4714). IEEE.

Koga, A., Okuda, H., Tazaki, Y., Suzuki, T., Haraguchi, K., & Kang, Z. (2016). Realization of different driving characteristics for autonomous vehicle by using model predictive control. In *2016 IEEE intelligent vehicles symposium (IV)* (pp. 722–728). IEEE.

Kohlbrecher, S., Von Stryk, O., Meyer, J., & Klingauf, U. (2011). A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE international symposium on safety, security, and rescue robotics* (pp. 155–160). IEEE.

Koukoumidis, E., Martonosi, M., & Peh, L.-S. (2011). Leveraging smartphone cameras for collaborative road advisories. *IEEE Transactions on Mobile Computing*, *11*(5), 707–723.

Kritayakirana, K., & Gerdes, J. C. (2012). *Autonomous vehicle control at the limits of handling* (Ph.D. thesis), CA: Stanford University Stanford.

Lafuente-Arroyo, S., Salcedo-Sanz, S., Maldonado-Bascón, S., Portilla-Figueras, J. A., & López-Sastre, R. J. (2010). A decision support system for the automatic management of keep-clear signs based on support vector machines and geographic information systems. *Expert Systems with Applications*, *37*(1), 767–773.

Larsson, F., & Felsberg, M. (2011). Using fourier descriptors and spatial models for traffic sign recognition. In *Scandinavian conference on image analysis* (pp. 238–249). Springer.

Laurense, V. A., Goh, J. Y., & Gerdes, J. C. (2017). Path-tracking for autonomous vehicles at the limit of friction. In *2017 American control conference (ACC)* (pp. 5586–5591). IEEE.

LaValle, S. M., & Kuffner Jr, J. J. (2001). Randomized kinodynamic planning. *The International Journal of Robotics Research*, *20*(5), 378–400.

Lee, U., Jung, J., Jung, S., & Shim, D. H. (2018). Development of a self-driving car that can handle the adverse weather. *International Journal of Automotive Technology*, *19*(1), 191–197.

Lee, S., Kim, J., Yoon, J. S., Shin, S., Bailo, O., Kim, N., Lee, T.-H., Hong, H. S., Han, S.-H., & Kweon, I. S. (2017). Vpgnet: Vanishing point guided network for lane and road marking detection and recognition. In *2017 IEEE international conference on computer vision (ICCV)* (pp. 1965–1973). IEEE.

Leedy, B. M., Putney, J. S., Bauman, C., Cacciola, S., Webster, J. M., & Reinholtz, C. F. (2007). Virginia tech's twin contenders: A comparative study of reactive and deliberative navigation. In *The 2005 DARPA grand challenge* (pp. 155–182). Springer.

Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., Kolter, J. Z., Langer, D., Pink, O., & Pratt, V. (2011). Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE intelligent vehicles symposium (IV)* (pp. 163–168). IEEE.

Levinson, J., & Thrun, S. (2010). Robust vehicle localization in urban environments using probabilistic maps. In *2010 IEEE international conference on robotics and automation* (pp. 4372–4378). IEEE.

Li, X., Sun, Z., Cao, D., He, Z., & Zhu, Q. (2015). Real-time trajectory planning for autonomous urban driving: Framework, algorithms, and verifications. *IEEE/ASME Transactions on Mechatronics*, *21*(2), 740–753.

Li, X., Sun, Z., Cao, D., Liu, D., & He, H. (2017). Development of a new integrated local trajectory planning and tracking control framework for autonomous ground vehicles. *Mechanical Systems and Signal Processing*, *87*, 118–137.

Lindner, F., Kressel, U., & Kaelberer, S. (2004). Robust recognition of traffic signals. In *IEEE intelligent vehicles symposium, 2004* (pp. 49–53). IEEE.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. In *European conference on computer vision* (pp. 21–37). Springer.

Lyrio, L. J., Oliveira-Santos, T., Badue, C., & De Souza, A. F. (2015). Image-based mapping, global localization and position tracking using vg-ram weightless neural networks. In *2015 IEEE international conference on robotics and automation (ICRA)* (pp. 3603–3610). IEEE.

Lyrio, L. J., Oliveira-Santos, T., Forechi, A., Veronese, L., Badue, C., & De Souza, A. F. (2014). Image-based global localization using vg-ram weightless neural networks. In *2014 international joint conference on neural networks (IJCNN)* (pp. 3363–3370). IEEE.

Massera Filho, C., Wolf, D. F., Grassi, V., & Osório, F. S. (2014). Longitudinal and lateral control for autonomous ground vehicles. In *2014 IEEE intelligent vehicles symposium proceedings* (pp. 588–593). IEEE.

Mathias, M., Timofte, R., Benenson, R., & Van Gool, L. (2013). Traffic sign recognition—How far are we from the solution?. In *The 2013 international joint conference on neural networks (IJCNN)* (pp. 1–8). IEEE.

McCall, J., & Trivedi, M. (2006). Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation. *IEEE Transactions on Intelligent Transportation Systems*, *1*(7), 20–37.

McNaughton, M., Urmson, C., Dolan, J. M., & Lee, J.-W. (2011). Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *2011 IEEE international conference on robotics and automation* (pp. 4889–4895). IEEE.

Mertz, C., Navarro-Serment, L. E., MacLachlan, R., Rybski, P., Steinfeld, A., Suppé, A., Urmson, C., Vandapel, N., Hebert, M., & Thorpe, C. (2013). Moving object detection with laser scanners. *Journal of Field Robotics*, *30*(1), 17–43.

Milford, M. J., & Wyeth, G. F. (2012). Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *2012 IEEE international conference on robotics and automation* (pp. 1643–1649). IEEE.

Miller, I., Campbell, M., Huttenlocher, D., Kline, F.-R., Nathan, A., Lupashin, S., Catlin, J., Schimpf, B., Moran, P., & Zych, N. (2008). Team cornell's skynet: Robust perception and planning in an urban environment. *Journal of Field Robotics*, *25*(8), 493–527.

Mnih, V., & Hinton, G. E. (2010). Learning to detect roads in high-resolution aerial images. In *European conference on computer vision* (pp. 210–223). Springer.

Mogelmose, A., Trivedi, M. M., & Moeslund, T. B. (2012). Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. *IEEE Transactions on Intelligent Transportation Systems*, *13*(4), 1484–1497.

Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., & Huhnke, B. (2008). Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, *25*(9), 569–597.

Moravec, H., & Elfes, A. (1985). High resolution maps from wide angle sonar. In *Proceedings. 1985 IEEE international conference on robotics and automation, Vol. 2* (pp. 116–121). IEEE.

Mouhagir, H., Cherfaoui, V., Talj, R., Aioun, F., & Guillemard, F. (2017). Trajectory planning for autonomous vehicle in uncertain environment using evidential grid. *IFAC-PapersOnLine*, *50*(1), 12545–12550.

Mouhagir, H., Talj, R., Cherfaoui, V., Aioun, F., & Guillemard, F. (2016). Integrating safety distances with trajectory planning by modifying the occupancy grid for autonomous vehicle navigation. In *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)* (pp. 1114–1119). IEEE.

Mutz, F., Cardoso, V., Teixeira, T., Jesus, L. F., Golçalves, M. A., Guidolini, R., Oliveira, J., Badue, C., & De Souza, A. F. (2017). Following the leader using a tracking system based on pre-trained deep neural networks. In *2017 international joint conference on neural networks (IJCNN)* (pp. 4332–4339). IEEE.

Mutz, F., Veronese, L. P., Oliveira-Santos, T., De Aguiar, E., Cheein, F. A. A., & De Souza, A. F. (2016). Large-scale mapping in complex field scenarios using an autonomous car. *Expert Systems with Applications*, *46*, 439–462.

Na, K., Byun, J., Roh, M., & Seo, B. (2015). Roadplot-datmo: Moving object tracking and track fusion system using multiple sensors. In *2015 international conference on connected vehicles and expo (ICCVE)* (pp. 142–143). IEEE.

Nguyen, T.-N., Michaelis, B., Al-Hamadi, A., Tornow, M., & Meinecke, M.-M. (2011). Stereo-camera-based urban environment perception using occupancy grid and object tracking. *IEEE Transactions on Intelligent Transportation Systems*, *13*(1), 154–165.

Nothdurft, T., Hecker, P., Ohl, S., Saust, F., Maurer, M., Reschka, A., & Böhmer, J. R. (2011). Stadtpilot: First fully autonomous test drives in urban traffic. In *2011 14th international IEEE conference on intelligent transportation systems (ITSC)* (pp. 919–924). IEEE.

Okumura, B., James, M. R., Kanzawa, Y., Derry, M., Sakai, K., Nishi, T., & Prokhorov, D. (2016). Challenges in perception and decision making for intelligent automotive vehicles: A case study. *IEEE Transactions on Intelligent Vehicles*, *1*(1), 20–32.

Oliveira, G. L., Radwan, N., Burgard, W., & Brox, T. (2017). Topometric localization with deep learning. arXiv preprint arXiv:1706.08775.

Omachi, M., & Omachi, S. (2010). Detection of traffic light using structural information. In *IEEE 10th international conference on signal processing proceedings* (pp. 809–812). IEEE.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, *9*(1), 62–66.

Overett, G., & Petersson, L. (2011). Large scale sign detection using hog feature variants. In *2011 IEEE intelligent vehicles symposium (IV)* (pp. 326–331). IEEE.

Paden, B., Čáp, M., Yong, S. Z., Yershov, D., & Frazzoli, E. (2016). A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, *1*(1), 33–55.

de Paula Veronese, L., Junior, L. J. L., Mutz, F. W., de Oliveira Neto, J., Azevedo, V. B., Berger, M., De Souza, A. F., & Badue, C. (2012). Stereo matching with vg-ram weightless neural networks. In *2012 12th international conference on intelligent systems design and applications (ISDA)* (pp. 309–314). IEEE.

Petrovskaya, A., Perrollaz, M., Oliveira, L., Spinello, L., Triebel, R., Makris, A., Yoder, J.-D., Laugier, C., Nunes, U., & Bessiere, P. (2012). Awareness of road scene participants for autonomous driving. *Handbook of Intelligent Vehicles*, 1383–1432.

Petrovskaya, A., & Thrun, S. (2009). Model based vehicle detection and tracking for autonomous urban driving. *Autonomous Robots*, *26*(2–3), 123–139.

Pettersson, N., Petersson, L., & Andersson, L. (2008). The histogram feature-a resource-efficient weak classifier. In *2008 IEEE intelligent vehicles symposium* (pp. 678–683). IEEE.

Pivtoraiko, M., Knepper, R. A., & Kelly, A. (2009). Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, *26*(3), 308–333.

Possatti, L. C., Guidolini, R., Cardoso, V. B., Berriel, R. F., Paixão, T. M., Badue, C., De Souza, A. F., & Oliveira-Santos, T. (2019). Traffic light recognition using deep learning and prior maps for autonomous cars. In *2019 international joint conference on neural networks (IJCNN)*. IEEE.

Radaelli, R. R., Badue, C., Gonçalves, M. A., Oliveira-Santos, T., & De Souza, A. F. (2014). A motion planner for car-like robots based on rapidly-exploring random trees. In *Ibero-American conference on artificial intelligence* (pp. 469–480). Springer.

Radwan, N., Tipaldi, G. D., Spinello, L., & Burgard, W. (2016). Do you see the bakery? leveraging geo-referenced texts for global localization in public maps. In *2016 IEEE international conference on robotics and automation (ICRA)* (pp. 4837–4842). IEEE.

Ramm, F., Topf, J., & Chilton, S. (2011). *OpenStreetMap: using and enhancing the free map of the world*. UIT Cambridge Cambridge.

Ramos, F., & Ott, L. (2016). Hilbert maps: scalable continuous occupancy mapping with stochastic gradient descent. *International Journal of Robotics Research*, *35*(14), 1717–1730.

Redmon, J., & Farhadi, A. (2017). Yolo9000: Better, faster, stronger. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 6517–6525). IEEE.

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (pp. 91–99).

Rohde, J., Jatzkowski, I., Mielenz, H., & Zöllner, J. M. (2016). Vehicle pose estimation in cluttered urban environments using multilayer adaptive monte carlo localization. In *2016 19th international conference on information fusion (FUSION)* (pp. 1774–1779). IEEE.

Sabbagh, V. B., Freitas, E. J., Castro, G. M., Santos, M. M., Baleeiro, M. F., da Silva, T. M., Iscold, P., Torres, L. A., & Pereira, G. A. (2010). Desenvolvimento de um sistema de controle para um carro de passeio autônomo. In *XVIII congresso brasileiro de automática*.

SAE (2018). *Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles*: *Technical report*, Warrendale, PA, USA: SAE International.

Samson, C. (1995). Control of chained systems application to path following and time-varying point-stabilization of mobile robots. *IEEE Transactions on Automatic Control*, *40*(1), 64–77.

Sarcinelli, R., Guidolini, R., Cardoso, V. B., Paixão, T. M., Berriel, R. F., Azevedo, P., De Souza, A. F., Badue, C., & Oliveira-Santos, T. (2019). Handling pedestrians in self-driving cars using image tracking and alternative path generation with frenét frames. *Computers & Graphics*.

Schaefer, A., Luft, L., & Burgard, W. (2018). Dct maps: Compact differentiable lidar maps based on the cosine transform. *IEEE Robotics and Automation Letters*, *3*(2), 1002–1009.

Schneider, F. E., & Wildermuth, D. (2011). Results of the european land robot trial and their usability for benchmarking outdoor robot systems. In *Conference towards autonomous robotic systems* (pp. 408–409). Springer.

Segal, A., Haehnel, D., & Thrun, S. (2009). Generalized-ICP. In *Robotics: Science and systems, Vol. 5* (168–176). Seattle, WA.

Sermanet, P., Eigen, D., Mathieu, M., Zhang, X., Fergus, R., & Lecun, Y. (2013). Overfeat detection using deep learning. In *International conference on learning representations (ICLR), Vol. 16*.

Shinzato, P. Y., dos Santos, T. C., Rosero, L. A., Ridel, D. A., Massera, C. M., Alencar, F., Batista, M. P., Hata, A. Y., Osório, F. S., & Wolf, D. F. (2016). Carina dataset: An emerging-country urban scenario benchmark for road detection systems. In *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)* (pp. 41–46). IEEE.

Sooksatra, S., & Kondo, T. (2014). Red traffic light detection using fast radial symmetry transform. In *2014 11th international conference on electrical engineering/electronics, computer, telecommunications and information technology (ECTI-CON)* (pp. 1–6). IEEE.

Spangenberg, R., Goehring, D., & Rojas, R. (2016). Pole-based localization for autonomous vehicles in urban scenarios. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 2161–2166). IEEE.

SparkFun (2018). Autonomous vehicle competition. https://avc.sparkfun.com/. Accessed: 22- May- 2018.

Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (2012). Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, *32*, 323–332.

Stentz, A., Fox, D., & Montemerlo, M. (2003). Fastslam: A factored solution to the simultaneous localization and mapping problem with unknown data association. In *In proceedings of the AAAI national conference on artificial intelligence*. Citeseer.

Suhr, J. K., Jang, J., Min, D., & Jung, H. G. (2016). Sensor fusion-based low-cost vehicle localization system for complex urban environments. *IEEE Transactions on Intelligent Transportation Systems*, *18*(5), 1078–1086.

Teixeira, T., Mutz, F., Cardoso, V. B., Veronese, L., Badue, C., Oliveira-Santos, T., & De Souza, A. F. (2018). Map memorization and forgetting in the iara autonomous car. arXiv preprint arXiv:1810.02355.

Thorpe, C., Herbert, M., Kanade, T., & Shafter, S. (1991). Toward autonomous driving: the cmu navlab. ii. architecture and systems. *IEEE Expert*, *6*(4), 44–52.

Thrun, S. (2010). Toward robotic cars. *Communications of the ACM*, *53*(4), 99–106.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. MIT press.

Thrun, S., & Montemerlo, M. (2006). The graph slam algorithm with applications to large-scale mapping of urban structures. *International Journal of Robotics Research*, *25*(5–6), 403–429.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., & Hoffmann, G. (2006). Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, *23*(9), 661–692.

Torres, L. T., Paixão, T. M., Berriel, R. F., De Souza, A. F., Badue, C., Sebe, N., & Oliveira-Santos, T. (2019). Effortless deep training for traffic sign detection using templates and arbitrary natural images. In *2019 international joint conference on neural networks (IJCNN)*. IEEE.

Trehard, G., Pollard, E., Bradai, B., & Nashashibi, F. (2014). Tracking both pose and status of a traffic light via an interacting multiple model filter. In *17th international conference on information fusion (FUSION)* (pp. 1–7). IEEE.

Ulbrich, S., & Maurer, M. (2013). Probabilistic online pomdp decision making for lane changes in fully automated driving. In *16th international IEEE conference on intelligent transportation systems (ITSC 2013)* (pp. 2063–2067). IEEE.

Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M., Dolan, J., Duggins, D., Galatali, T., & Geyer, C. (2008). Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, *25*(8), 425–466.

Veronese, L., Aguiar, E., Nascimento, R. C., Guivant, J., Cheein, F. A. A., De Souza, A. F., & Oliveira-Santos, T. (2015). Re-emission and satellite aerial maps applied to vehicle localization on urban environments. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 4285–4290). IEEE.

Veronese, L., Guivant, J., Cheein, F. A. A., Oliveira-Santos, T., Mutz, F., de Aguiar, E., Badue, C., & De Souza, A. F. (2016). A light-weight yet accurate localization system for autonomous cars in large-scale and complex environments. In *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)* (pp. 520–525). IEEE.

Viswanathan, A., Pires, B. R., & Huber, D. (2016). Vision-based robot localization across seasons and in remote locations. In *2016 IEEE international conference on robotics and automation (ICRA)* (pp. 4815–4821). IEEE.

Vivacqua, R., Vassallo, R., & Martins, F. (2017). A low cost sensors approach for accurate vehicle localization and autonomous driving application. *Sensors*, *17*(10), 2359.

Vu, T.-D., & Aycard, O. (2009). Laser-based detection and tracking moving objects using data-driven markov chain monte carlo. In *2009 IEEE international conference on robotics and automation* (pp. 3800–3806). IEEE.

Wang, D. Z., Posner, I., & Newman, P. (2015). Model-free detection and tracking of dynamic objects with 2d lidar. *International Journal of Robotics Research*, *34*(7), 1039–1063.

Wegner, J. D., Montoya-Zegarra, J. A., & Schindler, K. (2015). Road networks as collections of minimum cost paths. *ISPRS Journal of Photogrammetry and Remote Sensing*, *108*, 128–137.

Wei, J., Snider, J. M., Kim, J., Dolan, J. M., Rajkumar, R., & Litkouhi, B. (2013). Towards a viable autonomous driving research platform. In *2013 IEEE intelligent vehicles symposium (IV)* (pp. 763–770). IEEE.

Wolcott, R. W., & Eustice, R. M. (2017). Robust lidar localization using multiresolution gaussian mixture maps for autonomous driving. *International Journal of Robotics Research*, *36*(3), 292–319.

Wray, K. H., Witwicki, S. J., & Zilberstein, S. (2017). Online decision-making for scalable autonomous systems. In *International joint conference on artificial intelligence*.

Wu, T., & Ranganathan, A. (2012). A practical system for road marking detection and recognition. In *2012 IEEE intelligent vehicles symposium* (pp. 25–30). IEEE.

Xin, J., Wang, C., Zhang, Z., & Zheng, N. (2014). China future challenge: Beyond the intelligent vehicle. *IEEE Intelligent Transportation Systems Society Newsletter*, *16*(2), 8–10.

Xu, Y., John, V., Mita, S., Tehrani, H., Ishimaru, K., & Nishino, S. (2017). 3d point cloud map based vehicle localization using stereo camera. In *2017 IEEE intelligent vehicles symposium (IV)* (pp. 487–492). IEEE.

Xu, W., Snider, J., Wei, J., & Dolan, J. M. (2015). Context-aware tracking of moving objects for distance keeping. In *2015 IEEE intelligent vehicles symposium (IV)* (pp. 1380–1385). IEEE.

Xu, W., Wei, J., Dolan, J. M., Zhao, H., & Zha, H. (2012). A real-time motion planner with trajectory optimization for autonomous vehicles. In *2012 IEEE international conference on robotics and automation* (pp. 2061–2067). IEEE.

Xue, J.-r., Wang, D., Du, S.-y., Cui, D.-x., Huang, Y., & Zheng, N.-n. (2017). A vision-centered multi-sensor fusing approach to self-localization and obstacle perception for robotic cars. *Frontiers of Information Technology & Electronic Engineering*, *18*(1), 122–138.

Yenikaya, S., Yenikaya, G., & Düven, E. (2013). Keeping the vehicle on the road: A survey on on-road lane detection systems. *ACM Computing Surveys*, *46*(1), 1–43.

Yoon, S., Yoon, S.-E., Lee, U., & Shim, D. H. (2015). Recursive path planning using reduced states for car-like vehicles on grid maps. *IEEE Transactions on Intelligent Transportation Systems*, *16*(5), 2797–2813.

Yu, F., Xian, W., Chen, Y., Liu, F., Liao, M., Madhavan, V., & Darrell, T. (2018). Bdd100k: A diverse driving video database with scalable annotation tooling. arXiv preprint arXiv:1805.04687.

Zhang, L., Li, Q., Li, M., Mao, Q., & Nüchter, A. (2013). Multiple vehicle-like target tracking based on the velodyne lidar. *IFAC Proceedings Volumes*, *46*(10), 126–131.

Zhang, Y., Xue, J., Zhang, G., Zhang, Y., & Zheng, N. (2014). A multi-feature fusion based traffic light recognition algorithm for intelligent vehicles. In *Proceedings of the 33rd Chinese control conference* (pp. 4924–4929). IEEE.

Zhao, P., Chen, J., Song, Y., Tao, X., Xu, T., & Mei, T. (2012). Design of a control system for an autonomous vehicle based on adaptive-pid. *International Journal of Advanced Robotic Systems*, *9*(2), 44.

Zhao, L., Ichise, R., Liu, Z., Mita, S., & Sasaki, Y. (2017). Ontology-based driving decision making: A feasibility study at uncontrolled intersections. *IEICE Transactions on Information and Systems*, *100*(7), 1425–1439.

Zhao, L., Ichise, R., Yoshikawa, T., Naito, T., Kakinami, T., & Sasaki, Y. (2015). Ontology-based decision making on uncontrolled intersections and narrow roads. In *2015 IEEE intelligent vehicles symposium (IV)* (pp. 83–88). IEEE.

Zhu, Z., Liang, D., Zhang, S., Huang, X., Li, B., & Hu, S. (2016). Traffic-sign detection and classification in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2110–2118).

Ziegler, J., Bender, P., Dang, T., & Stiller, C. (2014). Trajectory planning for bertha—A local, continuous method. In *2014 IEEE intelligent vehicles symposium* (pp. 450–457). IEEE.

Ziegler, J., Bender, P., Schreiber, M., Lategahn, H., Strauss, T., Stiller, C., Dang, T., Franke, U., Appenrodt, N., & Keller, C. G. (2014). Making bertha drive—An autonomous journey on a historic route. *IEEE Intelligent Transportation Systems Magazine*, *6*(2), 8–20.

Ziegler, J., Lategahn, H., Schreiber, M., Keller, C. G., Knöppel, C., Hipp, J., Haueis, M., & Stiller, C. (2014). Video based localization for bertha. In *2014 IEEE intelligent vehicles symposium proceedings* (pp. 1231–1238). IEEE.

Ziegler, J., Werling, M., & Schroder, J. (2008). Navigating car-like robots in unstructured environments using an obstacle sensitive cost function. In *2008 IEEE intelligent vehicles symposium* (pp. 787–791). IEEE.

Zoller, E. (2018). *Location platform index: Mapping and navigation, 1H18: Key vendor rankings and market trends*: *Technical report*, Ovum - TMT Intelligence - Informa.