Integrated Planner for Autonomous Driving in Urban Environments Including Driving Intention Estimation

Hatem Darweesh

Nagoya University

Graduate School of Information Science, Department of Media Science

Doctoral Dissertation

Integrated Planner for Autonomous Driving in Urban Environments Including Driving Intention Estimation

by

Hatem Darweesh

Submitted to the Department of Media Science, Graduate School of Information Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science
January 2020

Abstract

Thousands are killed every day in traffic accidents, and drivers are mostly to blame. Autonomous driving technology is the ultimate technological solution to this problem. Moreover, autonomous driving technologies provide new social experience, reduce carbon emissions, reduce fuel consumption, save time and support aging communities. Planning, which determines the movement of autonomous vehicles, is the cornerstone of autonomous agent navigation. It is also one of the most difficult tasks to perform, because the safety of the vehicle, its passengers and other road users depend upon it.

Despite the thousands of autonomous vehicles currently operating on public roads for testing, there are still many unresolved problems with autonomous driving technology. For example, these systems are designed to drive defensively, but safety threats occur continuously while moving in traffic, especially in complex situations that involve interaction with human drivers. Better, more innovative, solutions for autonomous understanding of the intentions of other road users are still needed in order to achieve more natural driving performance. Another problem is that autonomous driving technology is still not standardized. Roads need to be mapped and driven on multiple times before automated driving tests can be conducted, and every development team has its own specific, closed implementations. This portability problem is the result of poor utilization of road network mapping standards.

Planning applications consist of multiple modules with different input/output spaces, and these modules need to be integrated correctly. Another challenge is the lack of open-source planning projects that allow cooperation between development teams globally. Problems such as social interaction, understanding the intentions of other road users, and environment differences between various countries cannot be solved by just one team of developers, no matter how resourceful they are.

Open-source autonomous driving planners should also meet certain standards, such as the ability to support multiple platforms and adherence to mapping standards, as well as having acceptable performance, usability and extensibility. Although several open-source motion planners are currently available, unfortunately they all have drawbacks, such as poor utilization of standard road network maps, inability to support multiple platforms, difficulty of use and customization, and a lack of tutorials and support.

One reason that existing solutions for global planning (i.e., point-to-point navigation) are not efficient or portable enough is that they do not use standard road network maps. Likewise, performance of local planning algorithms tends to be improved by using imprecise object representations, and intention and trajectory estimation solutions for surrounding vehicles are currently able to achieve acceptable results in custom situations, but they usually cannot be generalized for other driving scenarios. And current generalized intention estimation techniques are still unable to achieve acceptable performance.

In order to address these problems, in this dissertation we describe the develop-

ment of an open-source, integrated planner for autonomous navigation called **"Open-Planner"**. This planner is composed of a global path planner, a behavior planner, a probabilistic trajectory and intention predictor, and a local planner. The global planner generates smooth, global paths which are used as a reference, after considering traffic costs annotated in a road map. A road network map and a goal location are required to compute a global path and then execute it while avoiding obstacles. The local planner generates smooth, obstacle-free local trajectories, which are then used by a trajectory tracker to achieve low-level control. The behavior state generator then uses surrounding vehicle estimated intentions to handle tasks such as object following, obstacle avoidance, emergency stopping, stopping at stop signs and traffic light negotiation.

A novel technique for estimating the intention and trajectory probabilities of surrounding vehicles is also introduced, which enables long-term planning and reliable decision making. First, the behavior planner models an average driver following the driving rules, utilizing information provided by a road network map to provide the agent control signal. Next, a customized particle filter is integrated with the planner to model the uncertainty of various intentions and trajectories. This probabilistic filter uses multiple sensing cues, such as pose, velocity, acceleration and turn signal information. The proposed estimation method supports various sensor modalities, depending on the availability of different types of sensing information. Finally, by using the sensor data, the probabilistic process is able to estimate the probabilities of various trajectories and intentions.

The integrated planner described in this dissertation was evaluated using simulation, and through field experimentation with a non-holonomic, Ackerman steering-based mobile robot. Results from our simulation and field experimentation indicate that the proposed planner can generate global and local paths dynamically, navigate smoothly through highly dynamic environments and operate reliably in real time, by utilizing the probabilities of the estimated intentions and trajectories of other road users.

This integrated planner has already been implemented as part of Autoware, which is an open-source autonomous driving framework, built using the Robot Operating System (ROS). Autoware has drawn a lot of attention internationally, and has been utilized in several projects by academic and commercial teams. Collaboration and feedback from the open-source autonomous driving community has allowed us to tackle some of the problems mentioned above.

Acknowledgments

This research was supported by the OPERA project of the Japan Science and Technology Agency and by Nagoya University.

I would like to thank everyone who facilitated this project by helping me organize my research and conduct the necessary experiments, and who provided me with the information that I needed to complete it. In particular, I would like to thank Professor Eijiro Takeuchi for his constant support throughout all of my work, and Professor Kazuya Takeda for his guidance and encouragement. I would also like to thank Professor Shinpei Kato for introducing me to the amazing team at Nagoya University.

I would also like to thank the Real-World Data Circulation Leaders Program and its professors, who helped me, enriched me with knowledge and inspired me with new ideas.

I would also like to thank my father, Professor Abdelhameed Darweesh, for pushing me to always pursue knowledge and to climb higher, no matter the obstacles, and of course my mother, the light that has guided me through my darkest moments, for saving me from giving up and guiding me towards achieving the impossible. Thank you very much.

Finally, I would like to thank all of my friends, family and colleagues for their support, and for their understanding when I was busy working and couldn't be with them. To my lab and gym mates Ekim and Jacob, thank you, and I hope this is not the end.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Thousands of people around the world die every day in traffic accidents [1]. According to a 2018 World Health Organization report, a road user dies every 23 seconds [2]. The report also notes that the number of traffic deaths world-wide has reached 1.35 million a year. Injuries suffered in traffic accidents are now the leading killer of people between the ages of 5 and 29. Besides deaths, tens of millions more are injured or disabled every year, with many suffering life-altering injuries with long-term effects. These losses take a huge toll on families and communities.

There are several causes for the high number of traffic fatalities and injuries; rapid urbanization, low safety standards, insufficient law enforcement, people driving while distracted, fatigued, or under the influence of drugs or alcohol, speeding and failure to wear seat-belts or helmets. Drivers are responsible for most of these accidents, as opposed to equipment failure or road hazards.

Many researchers are trying to tackle this problem by eliminating the human factor, using state of the art technology to replace human drivers with a collection of sensors, hardware and software, in order to achieve automated driving. Self-driving vehicles (autonomous vehicles) are close to becoming a reality. Interest in this field was sparked by the DARPA Challenge in 2005 [3] and DARPA Urban Challenge in 2007 [4]. However, after more than a decade of development autonomous driving (AD) is still far from achieving its ultimate automation objectives. Figure 1-1 shows the milestones for autonomous driving, starting with the classical 100% human driver

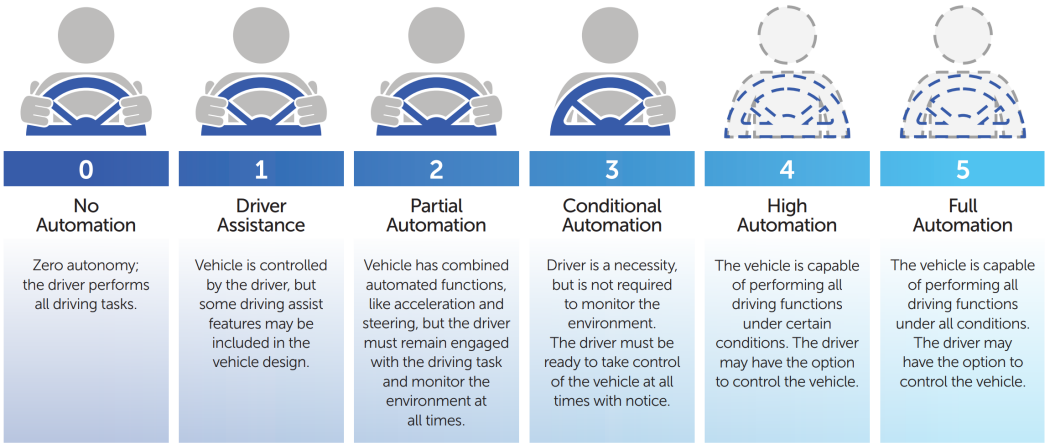


Figure 1-1: Automation levels and milestones for autonomous driving, as defined by the Society of Automotive Engineers (SAE).

(no automation) to futuristic, 100% computer automated systems (full automation) [5].

An autonomous vehicle is a combination of a drive by wire system, sensors, a computation platform and a collection of algorithms. Figure 1-2 shows Nagoya University's autonomous driving testing platform. This is the vehicle used to test the proposed integrated planner in a full-scale autonomous driving test. Autonomous driving still a difficult problem however, due to the vast number of possible situations that can occur in dynamic driving environments. In this work, I propose a solution to one of the most challenging autonomous driving tasks, which is planning. My research objective is to develop an open-source framework for planning which can achieve the following goals:

- Fosters international collaboration, so that different development teams can share feedback and solve problems using the same framework.
- Provides a complete autonomous driving software package, include all of the planning modules (global planner, local planner, intention prediction and behavior planner).
- Integrates the various planning modules so that they can be used as separate APIs or within a Robot Operating System (ROS).

Figure 1-2: sensor suite for Nagoya University Autonomous vehicle.

- Broad platform support, so that the system will work with wide range of autonomous platforms.
- Broad map support, so that the system will support standard and open-source map formats.
- Broad environment support, so that the system will work indoors, outdoors, on a wide variety of structured streets and off-road.
- Incorporates intention awareness, so that the system is able to predict the movement of other vehicles and plan accordingly.

1.1 Problem Definition

1.1.1 Planning as the brain for Autonomous Driving

Autonomous driving requires perception, localization, control and planning. Although there are currently many open-source resources available to researchers for perception, localization and control, there are few open-source planners that are general enough to be used directly, or which could be easily modified to suit a particular

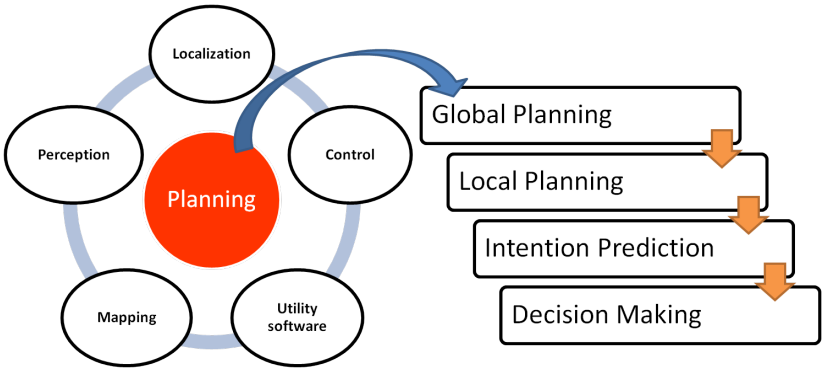


Figure 1-3: Level 4+ autonomous driving software stack.

application. This is because planning is the core module that connects everything together and it is also application dependent. Planning is also one of the most difficult autonomous driving tasks, as it determines the vehicle's actions, and the safety of the vehicle, its passengers and other road users depend on its correct operation.

Planning consists of multiple modules that need to be integrated together correctly, as shown in Figure 1-3. This section introduces autonomous driving planning challenges from different points of views, concentrating on the most difficult ones.

1.1.2 Social Interaction Challenges

For a very long time, autonomous vehicles will share the roads with human drivers, thus understanding the behavior of other drivers is one of the most challenging problems for autonomous vehicles to resolve. The ability to predict the intentions and trajectories of other vehicles is a very complex task, which even humans sometimes fail at, leading to traffic accidents.

Three words are often used interchangeably to express a driver or vehicle's actions; intention, behavior and maneuver [6]. 'Maneuver' is more specific than the other two and can include more than one action, especially during lateral changes of direction. 'Intention' and 'Behavior' are generally used to describe a driver's action, from doing nothing to the complicated, three-step passing of a slower vehicle. In this dissertation 'Behavior' is used to refer to the ego vehicle's actions or the estimated actions of other vehicles before filtering. 'Intention' is used for an estimated set of actions after

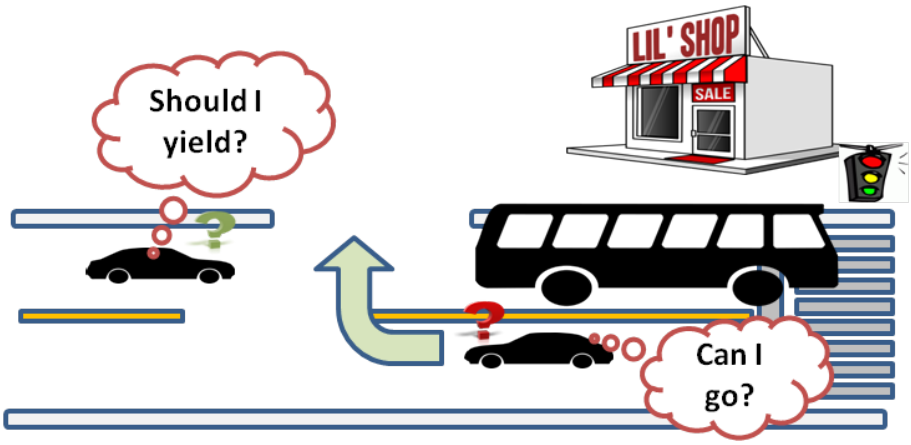


Figure 1-4: Vehicle turning across heavy oncoming traffic.

filtering. 'Intention Prediction' refers to guessing what another driver has in mind. Therefore, in this dissertation 'Intention' means driving behavior that is expected to be, or is currently being, executed by other vehicles.

The first common scenario is avoiding a traffic deadlock by letting a vehicle from the other side of the street enter an unregulated parking area. Figure 1-4 shows an intersection with a convenience store by the corner. The traffic light is red and a bus is stopped at the light. A vehicle traveling in the opposite direction wants to turn right to enter to the store's parking lot. If the vehicle following the bus does not yield, the turning vehicle will be blocked and will have to wait until another vehicle yields or until there are no vehicles coming from the opposite direction. This could take a while and might block the intersection.

A second common scenario is shown in Figure 1-5. A bus is stopping at a bus stop located on a two-way street. The vehicle behind the bus needs to figure out that this bus is stopping for a while. It has the choice of either waiting for the bus to move forward, or trying to pass the bus. The decision depends on whether or not there are vehicles coming from the opposite direction, and if so, on their intentions. The decision should also take into account what happens if the bus begins moving forward again (i.e., the intention of the bus driver), so that the passing maneuver can be aborted if necessary.

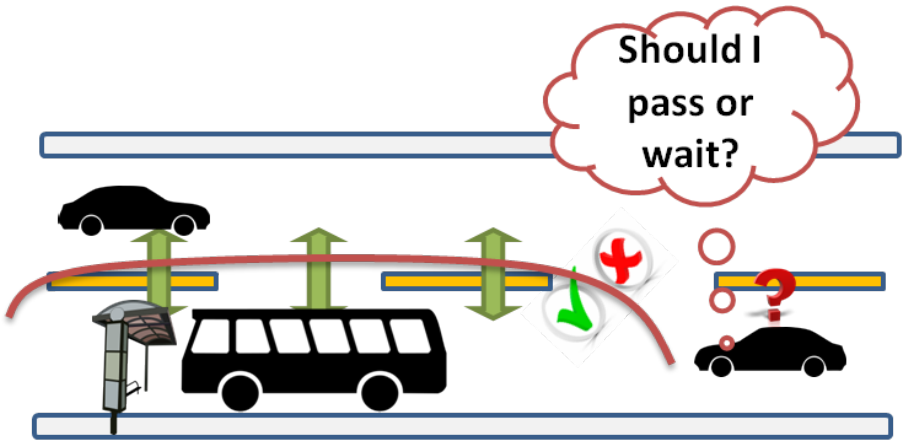


Figure 1-5: Passing a stopped vehicle on a two-way street.

1.1.3 Location and Culture Challenges

Although cars have been in common use for a long time, some aspects of driving differ from country to country, and even from city to city. We can summarize these differences as follows:

- In some countries vehicles travel on the left side of the road, in others vehicles travel on the right.
- Infrastructure differs from country to country and even from city to city in the same country. For example, the quality and design of roads, differences in signage and lane markings, use of traffic circles, etc.
- Driving rules and habits differ. For example, how close you can drive to the car in front of you, how close you can come to a vehicle when cutting back in after passing it, whether or not you need to stop for pedestrians waiting to cross at crosswalks that are not at an intersection, how far ahead of a turn you need to signal and which lane you should enter after making a cross-traffic turn onto a four-lane road.
- Social interaction rules are different, such as when to use the vehicle's horn, lights, hand motion, head motion, or even only eye contact to allow or forbid another road user from performing an action.

- Traffic laws are more strictly enforced in some countries than in others. What are traffic violations in some countries may be considered just rude behavior in others.

From the above examples, we can see that developing an autonomous driving planner that can handle all of these issues is an impossible task for one team. No single company or university can provide solutions to all of these problems. We believe solving these problems requires international cooperation between researchers and companies from around the globe, by sharing data, experience and solutions.

1.2 Basic Planning Components

To achieve full planning, pipeline inputs such as agent position, detected objects, maps, traffic information and goal location are needed. The output will be a smooth, obstacle free trajectory. The traditional approach is to develop multiple planners that solve the problem step by step, as shown in Figure 1-3. A global planner, local planner, intention predictor and behavior planner are all needed. Previously developed planners have tended to be either proprietary or open-source. Open-source planners are usually difficult to use, are designed for a specific environment or platform, or lack sufficient documentation or tutorials, as in [7][8][9].

1.2.1 Global Planner

The global planner module calculates the directed path that the vehicle should follow from start to goal. This process is also known as 'routing' or 'navigation', and an example of this process is shown in Figure 1-6. Global planning algorithms such as A* [10] and Anytime Dynamic A* [11] can solve global planning problems, but they usually require special customization and can only achieve marginal performance. The global planner proposed in this dissertation uses a road network map to represent the environment, and dynamic programming to search for a solution.

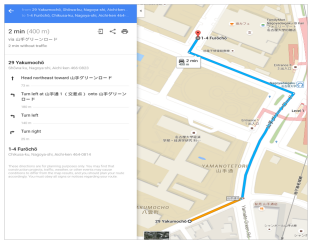


Figure 1-6: Navigation/Routing as an example a global planner.

1.2.2 Local Planner

The local planner module is responsible for selecting an obstacle free path from among multiple sampled paths. Examples of local planning results are shown in Figure 1-7. Sampling-based local path planning is a common method used to generate smooth trajectories, such as in [12][13]. Obstacle representation and trajectory selection are often bottlenecks, however. In Chapter 3 of this work, a clever method of representing detected obstacles is proposed, which improves both accuracy and performance.

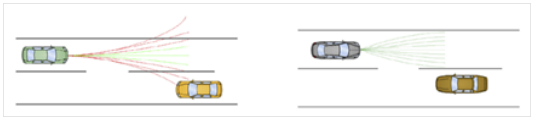


Figure 1-7: Local planner selecting the best path.

1.2.3 Intention prediction

It is important to estimate what other road users are currently doing, as well as their future trajectories, especially in complex scenarios such as the one shown in Figure 1-8. Solutions for this problem were proposed in [14][15], but there are problems with these solutions such as a lack of generalization for both driving scenarios and sensing models, as well as trade-offs between accuracy and performance. In this dissertation, I propose a behavior planner that can deal with many different driving situations. Performance of probabilistic filtering is also improved by reducing the dimensions of the state space.

Figure 1-8: Understanding other road users' intentions and likely trajectories is very useful in highly dynamic environments.

1.2.4 Behavior Planner

Also known as a 'decision maker', the behavior planner module selects the most appropriate action for the ego vehicle to execute. Examples of these actions are commands such as Forward, Follow, Change Lanes, Stop for Stop Sign, and Stop for Red Light, etc. The state machine shown in Figure 1-9 illustrates some of the behaviors that these kinds of planners generate. State machines are a simple and efficient way to represent such behaviors, as described in [16][17].

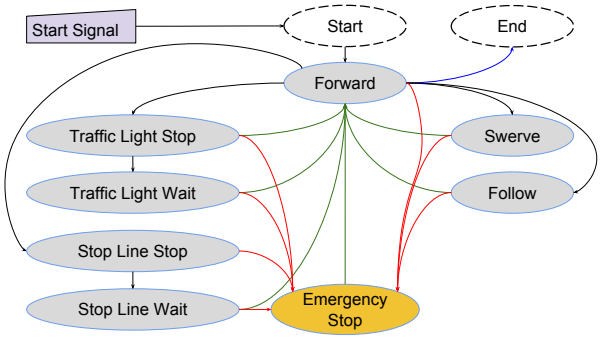


Figure 1-9: Behavior states are usually a connected graph of actions that the vehicle could execute.

1.3 Contributions

This section summarizes the contributions made by this dissertation, which are:

- Development of a complete, integrated open-source planner for autonomous

driving (Chapter 3).

- Introduction of a new technique for estimating the intentions and trajectories of surrounding vehicles (Chapter 4).

In addition to these two, main contributions, several other contributions were also introduced in this work:

- Utilization of road network maps.
- Use of flexible routing costs for global planning.
- Precise object representation to improve obstacle avoidance.
- Improvements to the performance of the particle filter.
- Increased flexibility, allowing additional types of sensing measurements.

Additional details are provided in Section 1.4.

1.4 Proposed Solution

In this section, the proposed integrated planner is introduced. Its success, based on its introduction as an open-source autonomous driving planner, is also discussed. A comparison of our planner and leading open-source planners is also provided.

1.4.1 OpenPlanner: An Open-Source, Integrated Planner

The implementation of the open-source, integrated planner introduced in this work is called OpenPlanner. Its architecture is illustrated in Figure 1-10. It includes a global planner that generates global reference paths using a vector (road network) map. The local planner then uses this global path to generate an obstacle-free, local trajectory from a set of sampled roll-outs. It uses various costs, such as collision, traffic rules, transition and distance from center, to select the optimal trajectory. An intention and trajectory estimator calculates the probabilities associated with other vehicles'

intentions and trajectories, while the behavior generator uses predefined traffic rules and sensor data to function as a decision maker.

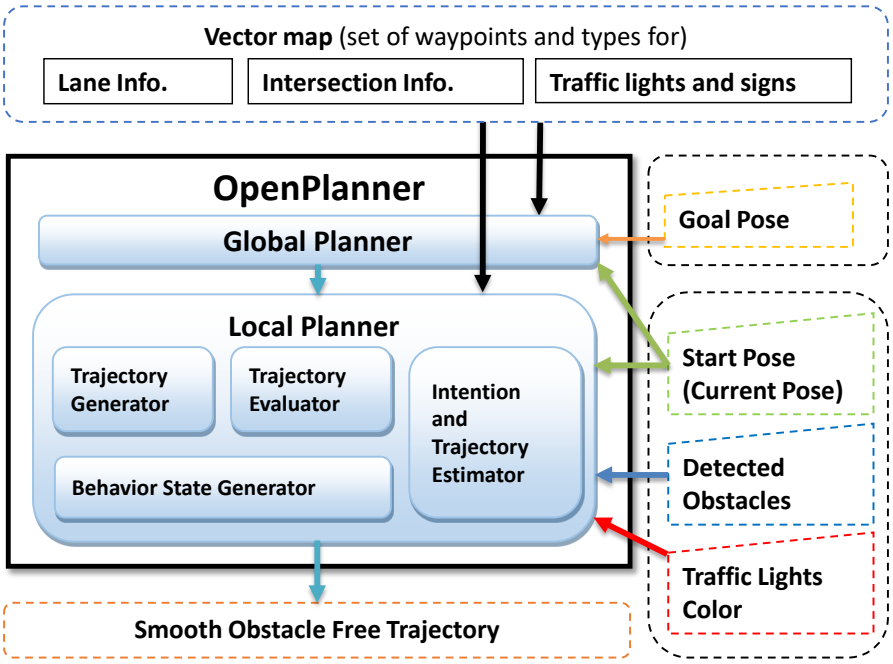


Figure 1-10: Architecture of the proposed integrated planner.

The research presented in this thesis was originally developed for autonomous driving applications. Its source code is included in the Autoware repository [18]. Autoware is an open-source, autonomous driving framework developed by Nagoya University, which is used by many researchers for autonomous driving research and development [19][20]. Autoware is based on the Robot Operating System (ROS) described in [21], and is a collection of ROS packages plus additional helper libraries. OpenPlanner will work with any mobile robot or vehicle by simply adjusting its parameters, and has been tested with both differential drive and non-holonomic robots.

OpenPlanner has been used by many international research teams, which have made diverse and innovative contributions by applying it to wide range of autonomous driving planning problems around the world, for example:

- **The Roboat project**: The objective of this project is to encourage the use of Amsterdam's canals for transportation, using a fleet of autonomous boats

[22][23].

- **Autonomous driving map editor**: road network map editor developed at the University of Konkuk, Korea [24].
- **ADAS Demo**: An Advanced Driver Assistance System (ADAS) demo for a major Hong Kong-based technology company.

Details of the contributions made through the development of OpenPlanner, which were summarized in Section 1.3, include:

- The utilization of road network maps improves planner portability, and enables the planner to handle more complex traffic scenarios.
- Use of flexible routing costs in the global planner allows integration with HMI modules, providing a seamless method of working with dynamic maps.
- Precise object representation using object contours instead of bounding boxes improves obstacle avoidance accuracy.

Table 1-11 compares OpenPlanner to the top open-source planners currently available. Moreover, our planner is being continuously improved thanks to feedback received from the open-source community.

1.4.2 Trajectory and Intention Estimation

We have developed a novel method of estimating the probabilities of the intentions and trajectories of surrounding vehicles, using an existing behavior planner with a particle filter. This estimation process is a very important function of the planner, allowing it to handle complex traffic situations. Figure 1-12 shows the estimator successfully predicting that the bus is going to stop at the bus stop before it fully executes the action. It is also able to estimate whether the following vehicle is yielding to the ego vehicle or passing it.

Details of the contributions of this work related to intention estimation, summarized in Section 1.3, are as follows:

Planner / Feature	Open Motion Planning Library **(OMPL)**	**ROS Navigation Stack**	Open Robotics Design & Control **Open-RDC**	Mobile Robot Programming Toolkit **MRPT**	**Apollo AD Planner**	**OpenPlanner**
Platform Support	Robots	Robots	Robots	Robots	Autonomous Vehicles	**Robots + Autonomous Vehicles**
Environment Support	Indoor	Indoor, Side-alk	Indoor, Sidewalk	Indoor	Pre-driven structured roads	**Indoor, outdoor, public road, highway**
Library APIs	Independent	-	-	Independent	-	**Independent**
ROS Support	Yes	Yes	Yes	Via bridge	-	**Yes**
Integration	-	-	-	-	Apollo	**Autoware**
Map Support	Cost map + point cloud	Cost map	Cost map	Cost map + point cloud	Customized OpenDRIVE	**Vector Map, KML, OpenDRIVE, Lanelet2**
Global Planning	Yes	-	-	-	Yes	**Yes**
Local Planning	Yes	Yes	Yes	Yes	Yes	**Yes**
Behavior Planning	Custom	-	-	-	Yes	**Yes**

Figure 1-11: Comparison of OpenPlanner with leading open-source planners.

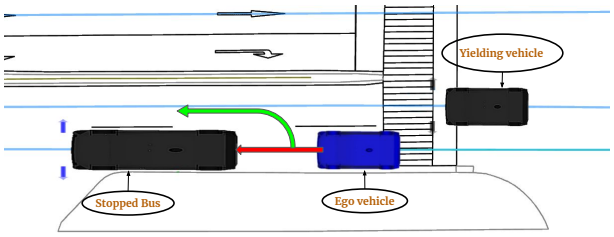


Figure 1-12: Bus stop situation.

- Utilization of an improved behavior planner allows the modeling of the internal parameters of the surrounding vehicles, which represent the behavior of an average driver.

- Use of a multi-cue particle filter reduces the dimensions of the state space within the particle filter, improving performance.

- Ability to use a variety of sensing measurements provides the flexibility of adding new sensing models or even disabling existing models.

1.5 Data Circulation and Social Impact

Data circulation, especially of real-world data, is essential for the development of autonomous driving applications. The concept of Real-World Data Circulation (RWDC) is utilized by the open-source integrated planner described in this dissertation. Data is generated from simulations, robot testing and real vehicle testing, and then shared with other teams for analysis. We also used other open data sets to test and improve our work. Figure 1-13 shows the RWDC concept as implemented for this project.

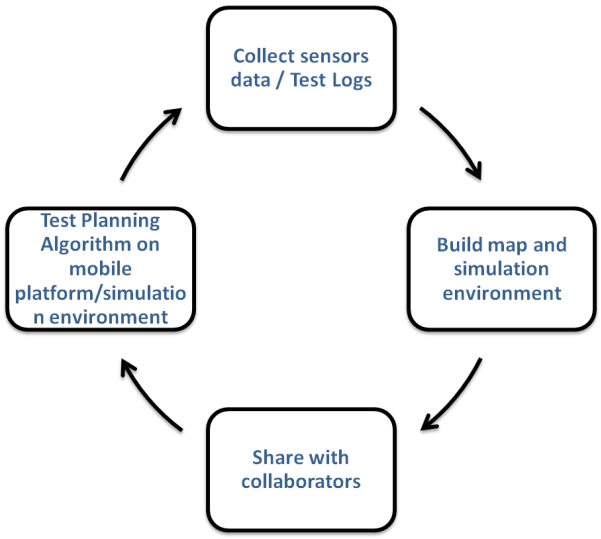


Figure 1-13: Relationship between OpenPlanner and RWDC. First sensor data is collected, then maps and simulation environment is built and shared with open-source community, then we develop and test planning algorithms, finally experiments data is shared with the development community.

Providing the integrated planner as open-source has contributed to another form of data circulation. It enabled teams to work with the same data formats, thus the data sharing is much easier. Teams around the world shared real data from their experiments, which enabled other teams to solve more challenging problems.

The social impact of developing the proposed integrated planner as an open-source project is demonstrated by the number of projects which are already utilizing it (see Section 1.4.1). In addition, the original paper in which this integrated planner was proposed [25] has been downloaded more than 3,000 times. Other research projects were also made easier by utilizing our planner. Furthermore, saving lives is the ulti-

mate objective of development of better planners for autonomous driving. Another social impact is demonstrated by the Roboat project [23], which is shifting transportation around Amsterdam from land to water, using a fleet of autonomous boats. In Chapter 5, we explain in more detail the concept of RWDC and the social value of the work presented in this dissertation.

1.6 Thesis Structure

In Chapter 2, we review the previous research that inspired our work. In Chapter 3, we explain in detail the components of OpenPlanner, with intention and trajectory estimation discussed at length in Chapter 4. The relationship between OpenPlanner and the real-world data circulation is explained in Chapter 5. Finally, in Chapter 6, this dissertation is concluded and future work is discussed. Figure 1-14 shows where each planning component is explained in the thesis.

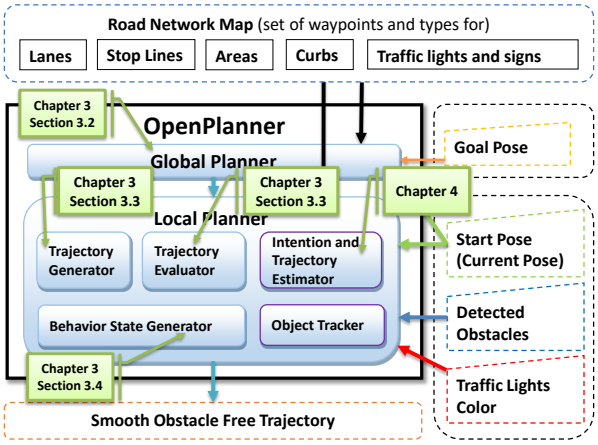


Figure 1-14: The integrated planner (OpenPlanner) is explained mainly in Chapters 3 and 4.

Chapter 2

Previous Work

Autonomous driving requires several levels of planning, including global planning, path planning, local planning and behavior planning. Museum tour guide robots are one example of autonomous navigation in indoor mobility [26]. Studies on long-range outdoor robot navigation [27][28] have shown that state-of-the-art planning techniques can achieve good results. Moreover, the well-known DARPA Urban Challenge has shown that robotic navigation of car-like vehicles operating in real traffic is feasible [16][12].

In robotics, planning is the task of finding a collision-free trajectory, from a starting position to a goal location, and thus it determines navigation decisions. Automated planning has been widely studied, from simple collision avoidance in simulated environments [29] to advanced algorithms which include vehicle constraints and uncertainty [30][31]. Planning for robot navigation usually involves the computation of global and local plans. Global planners compute paths from a current position to a goal location by satisfying optimal functions, usually using distance constraints as in Dijkstra [32]. Task planners function as an orchestrator by deciding when to start, stop, create a new plan, switch to an emergency state, and so on. In the following subsections we will cite examples of related work for each specific aspect of our proposed planning system.

2.1 Behavior Planning

The other important planning function besides path planning is task planning, which is also known as behavior generation. It generally uses a state machine to represent tasks and apply the rules that govern transitions between these tasks. In [17] researchers transformed a continuous driving behavior state into discrete state spaces, and then used a search algorithm to obtain the optimal task sequence to reach the goal condition in a symbolic space.

2.2 Global Planning

Some global path planners, such as A* [10], use heuristic functions, while others, such as Anytime dynamic A* [11] and the D* algorithm [33], employ re-planning in addition to heuristic functions. In [34], Hybrid A* is introduced, which combines three methods, heuristics, re-planning and a kinematically feasible search model, as shown in Figure 2-1. There are topological approaches as well, such as Voronoi graphs [35], which also compute collision free paths. These techniques are based on grid maps updated with sensor information, also known as cost maps. Such techniques create global planes for unstructured environments like off-road navigation and parking situations.

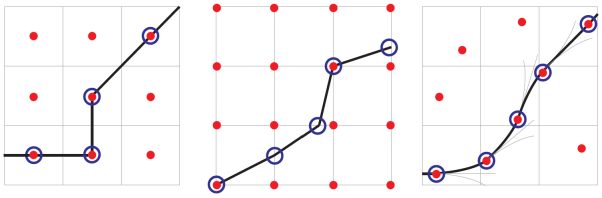


Figure 2-1: Graphical comparison of search algorithms. Left: A* associates costs with centers of cells and only visits states that correspond to grid-cell centers. Center: Field D* associates costs with cell corners and allows any linear path from cell to cell. Right: Hybrid A* associates a continuous state with each cell, and the score is the cost of its associated continuous state [34].

Networks of structured roads extending for several kilometers are another type of environment. With environments of this size, cost maps become impractical and a

different method of environment presentation is needed.

At the 2007 DARPA Urban Challenge, teams received a road network definition file (RNDF) for the complete course, which they used to globally plan their robot's motion through the challenge objectives, as described in [36][12]. Dynamic programming techniques for global planning were used by Stanford University's team, as described in [12], which involved dynamic programming with accelerating nodes. An example of map and planning results from [36] are shown in Figure 2-2.

Figure 2-2: Global planning: DP propagates values through a crude discrete version of the environment map. The color of the RNDF is representative of the cost to move to the goal from each position in the graph. Low costs are green, and high costs are red [36].

Clearly, this kind of structured environmental information made it practical for teams to plan global paths, which included lane changes, intersection negotiation, stop signs, traffic lights and parking. Since then, RNDFs have become essential for autonomous navigation. In [37], an optimized RNDF for autonomous driving was introduced. The type of RNDF maps used in this work are called vector maps. Although the structure of these maps is similar to open street maps [29], they are much more precise and include additional, regularly updated, information.

2.3 Local Planning

Several types of local planners have been proposed. Potential field approaches assign repulsive forces to obstacles and attractive forces to obstacle-free spaces [33]. Other successful obstacle avoidance algorithms consider vehicle constraints while predicting the future position of the vehicle [38]. The global dynamic window approach integrates global path information and uses it for obstacle avoidance [39][40].

More recently developed planners take human factors into account, in order to compute paths which are comfortable for passengers [41][42]. Efficient methods of local planning which generate multiple rollouts, starting from the center of a vehicle and running parallel to a reference path, have also been introduced [12][13]. These roll-outs are then linearly sampled and optimized to satisfy vehicle kinematics [36], as shown in Figure 2-3.

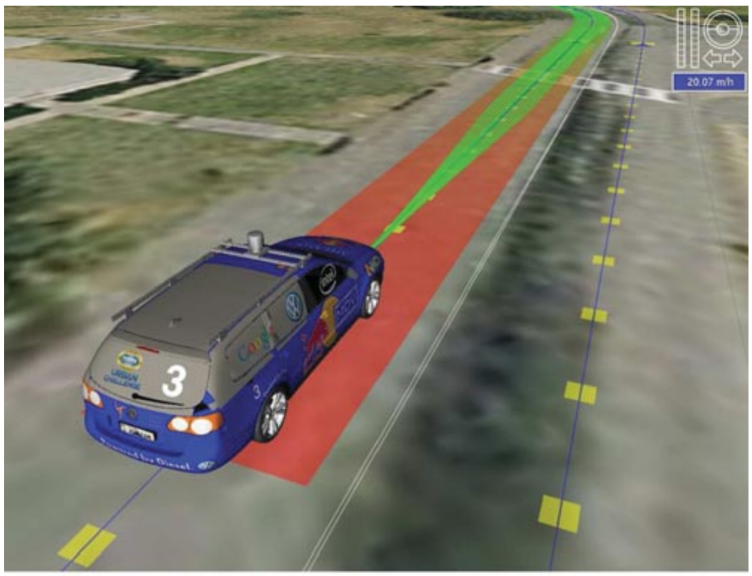


Figure 2-3: Planner rollouts in an urban setting, with multiple discrete choices [36].

Object detection and tracking is an important component of many local planning approaches, including our own. Noisy sensors, faulty detection algorithms and weather conditions contribute to false positive and false negative obstacle detection, while imprecise object representation methods, such as bounding boxes, also have

drawbacks. For example, from a detection point of view, a 20 cm error of margin is not a problem as long as the detection results are correct, but from planning perspective this 20 cm could lead to catastrophic results, as illustrated in Figure 2-4. It is essential that local planning methods have reliable object tracking capabilities, especially in outdoor, autonomous navigation applications. In [43], multiple hypothesis tracking (MHT) was used to achieve multiple target tracking, while other researchers have used probabilistic filters, such as Kalman or Particle filters.

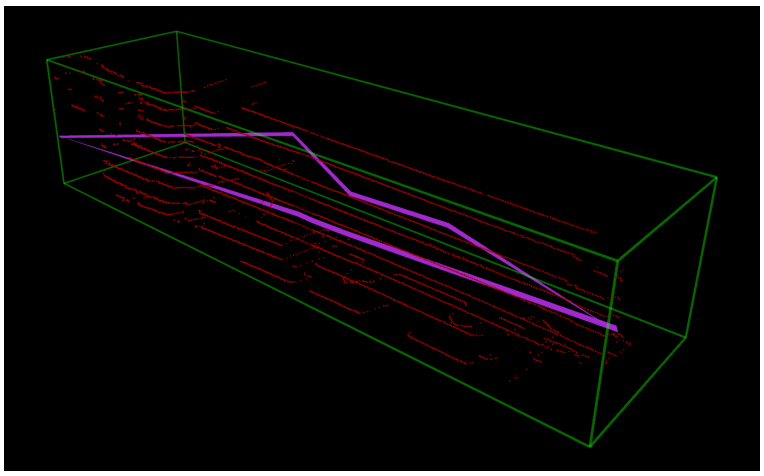


Figure 2-4: Red points represents the actual observed point cloud from LiDAR sensor data. The green box represents the bounding box. Pink lines represent the bounding contour. For objects with irregular shapes, bounding box error margins can become problematic.

2.4 Trajectory and Intention Estimation

The problem of estimating the behavior of other vehicles can be divided into two parts, trajectory estimation and intention estimation. Note that there is a difference in the state space between the two tasks; trajectories are continuous in an x, y, z, θ, v space [44][45], but intentions, such as going forward, turning, slowing down, yielding, etc., are discrete, and are usually modeled using a state machine, and solved using search-based algorithms [12][25]. Others have used one model for both trajectory and intention estimation however, as in [46][47].

One method of estimating the trajectories of other vehicles is to use deterministic models, as in [44]. This approach requires an accurate physical model of the moving object and the environment, and it can only estimate the trajectory a short time in the future, e.g., about one second. Another problem with this approach is that it does not account for observational uncertainties. Trajectories can also be estimated using machine learning, such as in [48][49][50]. Results using the machine learning approaches are good for predicting trajectories in specific situations, such as highway merging and exiting, at four-way intersections, etc. The main issue with this approach is difficulty providing sufficient driving scenario samples for training. Since some behaviors occur only rarely, machine learning approaches generalize these situations poorly.

2.4.1 Probabilistic Methods

To develop a successful trajectory and intention estimator for autonomous vehicles, two major uncertainties need to be addressed, observational uncertainty and motion uncertainty. One common way to model such uncertainty is by using a Dynamic Bayesian Network (DBN) [51]. Figure 2-5 shows a DBN with a state X, motion control signal u and observation Z.

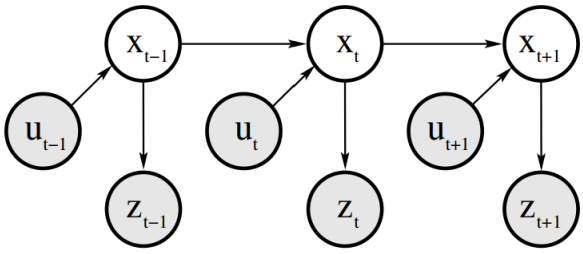


Figure 2-5: A dynamic Bayesian network characterizing the evolution of controls, states and measurements [51].

In [52], a driving behavior estimation and classification model is developed based on Hidden Markov Models (HMMs). HHMs are a popular solution for modeling time series data, representing probability distributions over a sequence of observations,

hence their use for estimating intention states [53]. Let's denote an observation at time t by the variable Z_t, with the assumption that the observations are sampled at discrete, equally-spaced time intervals. A hidden state, in this case driving intention X, at time t, is denoted by X_t, satisfying the Markovian property. Thus, the joint distribution of a sequence of states and observations can be factored as shown in Equation 2.1 [53]. This factorization of the joint probability is represented graphically in Figure 2-6.

$$P(X_{1:T}, Z_{1:T}) = P(X_1)P(Z_1|X_1)\prod_{t=2}^{T} P(X_t|X_{t-1})P(Z_t|X_t) \tag{2.1}$$

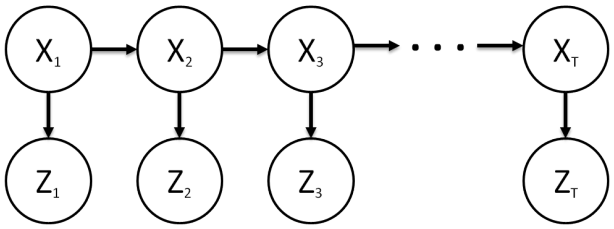


Figure 2-6: A Bayesian network specifying the conditionally independent relationships of a Hidden Markov Model [53].

Recently, there have been attempts to develop approaches that can address both trajectory and intention uncertainties in one step. In [54], an intention-aware planning application was proposed in which Partially Observable Markov Decision Processes (POMDP) were used to represent both trajectory and intention uncertainty, and the uncertainty problem was then solved using a Monte Carlo (MC) sampling method. In [14] POMDP was also used for decision making. Observations, i.e., the behaviors of other drivers, are usually modeled as a DBN, and then the whole system is solved using value iteration. In [14] the focus was only on modeling highway merging situations, and generalization to more complex driving situations was not addressed. Another POMDP model was introduced in [15], using a point-based solution. Thus, we can see that POMDP provides a rich framework for uncertainty modeling. When using POMDP, the model is represented by $X, A, Z, T, \hat{Z}, R, \gamma$, in which X repre-

sents the state, A represents actions or control decisions, Z represents observations, T represents the transition function between actions and states, $\hat{Z}$ is an observation function that connects the action and previous state to observations, R is a reward function deriving the system objectives, and γ is a discount factor between 0 and 1. The output of the system, in this case the estimated state, is called the belief state, denoted b. According to Bayes theorem, this belief state can be calculated as shown in Equation 2.2 [51].

$$b_z^a(\acute{x}) \propto \sum_x b(x)T(x,a,\acute{x})\hat{Z}(a,\acute{x},z) \tag{2.2}$$

Thus, the process model's state can be represented as $p(\acute{s}|s,a) = p(s_{t+1}|s_t,a)$. Knowledge about the system state can be represented by the belief distribution $b \in B$, and this belief can be updated after observing $z \in Z$ using observation model $p(z|\acute{s}) = p(z|s_{s+1})$. Value iteration is a common method used to solve discrete POMDPs. At each iteration, the system receives a reward $r : X \times A \rightarrow \Re$. The goal of the POMDP solver is to find a poly $\pi : B \rightarrow A$ which maximizes the value $V : B \rightarrow \Re$ for initial belief b_0. The value V is the expected sum of rewards for belief b_0 over time t, discounted by $\gamma \in [0,1)$, as shown in Equation 2.3:

$$V_\pi(b_0) = E\left[\sum_{t=0}^{\infty} \gamma^t r(x_t, \pi(b_t))\right] \tag{2.3}$$

Issues with MDP and POMDP approaches are the need to discretize the state space in order for the algorithm to become computationally tractable, but even this technique cannot produce real-time results. Beside the performance issues, all previous studies have concentrated on one type of traffic situation, without generalizing the model for use in other complex traffic scenarios. One source of the generalization problem is the use of primitive and custom lane-based road representations.

General Markovian filtering methods, such as the Kalman Filter (KF) and Particle Filter (PF), are known for their ability to model noise in data [51], allowing them to handle different types of uncertainty. Particle filters, due to their ability to approximate almost any distribution, are especially useful for modeling both continuous and discrete data. In [55], a particle filter was used with a dynamic vehicle model to estimate driver intention.

The basic idea behind particle filtering is to approximate a belief state with a set of weighted particles or samples [53], as shown in Equation 2.4. X_t is the state vector at time t, z is the observation, and X_t^i represents the i-th sample of X_t. Equation 2.4 allows us to compute the posterior using importance sampling. Because it is difficult to sample from the target distribution, we instead sample from proposal or importance distribution $q(x)$, then weight the samples, as shown in Equation 2.5:

$$P(X_t|z_{1:t}) \approx \sum_{i=1}^{N_s} w_t^i \delta(X_t, X_t^i) \tag{2.4}$$

$$w^i \propto \frac{P(x_{1:t}^i|z_{1:t})}{q(x_{1:t}^i|z_{1:t})} \tag{2.5}$$

One key feature of the particle filter is that the posterior is approximately represented by a set of particles, where each particle includes a state vector x and an associated weight w. The weights have to be normalized, for example: $\sum_i w_t^{(i)} = 1$.

When conventional particle filters are used to solve complex problems, such as estimating a vehicle's intention, they tend to perform very slowly due to the large dimensions of the state. Another problem is that the vehicle's state information is not mix of continuous and discrete data. One solution to these problems is to use a multi-cue particle filter [56]. Different sensing cues with variable levels of confidence could easily utilize the autonomous driving system's sensing modules, such as object detection, scene recognition and turn signal sensing.

For the estimation problem, the major uncertainty is observational uncertainty.

One example is human decision uncertainty, which affects the estimation of intention and occurs when a driver changes his mind suddenly, and goes left instead of right, for example. Other sources of uncertainty are the limitations of vehicle sensors and of the perception algorithms used to calculate the states of the surrounding vehicles. As a general rule, the more sensing information that is available, the more reliably algorithms can model the posterior distribution.

2.4.2 Utilization of Multiple Sensors

Multiple sensing cues were used to update a particle filter in [57][56]. Although it is a simple solution, it is effective for two reasons. First, having more information allows better design and modeling. Second, not all autonomous vehicles or robots are equipped with the same set of sensors, hence using a multi-cue particle filter allows better integration and a higher level of generalization.

2.4.3 Environment Representation

Environment representation is very important for state modeling. Some studies [50][57][58] have used very simple, custom-built road networks which provide basic driving rules in a simulation environment, but since they do not use a standard mapping format they are difficult to integrate with other autonomous driving systems.

2.5 Open-source planners

The two major open-source planners currently available are the Open Motion Planning Library (OMPL) [7] and Navigation Stack [8]. OMPL is basically a collection of APIs which can be used with or without ROS, while Navigation Stack is part of ROS and cannot be used outside it. Our implementation of OpenPlanner is similar to OMPL in that it is a collection of C++ APIs which can be used as a black box on a wide range of platforms. It also has ROS nodes that can be used directly with Autoware or any other ROS-based framework.

Open-rdc is open-source, robotic navigation programming software [9] based on the ROS Navigation Stack [8]. It can be used as an extension of Navigation Stack for applications involving differential-drive mobile robots. Open-rdc was developed for use at the Tsukuba Real World Robot Challenge (RWRC) in 2015. Another recently released integrated planner, known as the Apollo AD Planner, is part of the Apollo open-source autonomous driving framework [59]. Although it can handle most autonomous driving challenges, there are some problems with this planner. These include:

- Limited map support. Apollo AD Planner only supports a modified version of OpenDRIVE [60]. No open-source documentation is available for the modified map format.
- Difficult to use as a stand-alone library.
- Difficult to customize for mobile robot applications.

Figure 2-7 shows the capabilities of each open-source planner in regards to generally desirable features for planning frameworks. When developing OpenPlanner, we tried to address the drawbacks of other open-source planners, as much as possible.

Planner / Feature	Open Motion Planning Library **(OMPL)**	**ROS Navigation Stack**	Open Robotics Design & Control **Open-RDC**	Mobile Robot Programming Toolkit **MRPT**	**Apollo AD Planner**
Platform Support	Robots	Robots	Robots	Robots	Autonomous Vehicles
Environment Support	Indoor	Indoor, Side-alk	Indoor, Sidewalk	Indoor	Pre-driven structured roads
Library APIs	Independent	-	-	Independent	-
ROS Support	Yes	Yes	Yes	Via bridge	-
Integration	-	-	-	-	Apollo
Map Support	Cost map + point cloud	Cost map	Cost map	Cost map + point cloud	Customized OpenDRIVE
Global Planning	Yes	-	-	-	Yes
Local Planning	Yes	Yes	Yes	Yes	Yes
Behavior Planning	Custom	-	-	-	Yes

Figure 2-7: Comparison of major open-source planners. Apollo AD planner is the only planner that supports autonomous driving applications.

Chapter 3

Integrated Planner for Autonomous Navigation

3.1 Introduction

Although there are currently many open-source resources available to researchers for perception, localization and control, it is difficult to find an open-source planner that is general enough to be used directly or which can be easily modified to suit a particular application. For example, global and local planning are either tightly coupled or are developed totally separately. There are two main problems with tightly coupling global and local planning. First, they can't be used separately. The second problem is scalability, since it is not possible to use larger maps. Therefore, when developing OpenPlanner we employed the integrated approach, as shown in Figure 3-1. We developed separate global and local planning modules, but they share a standard interface which facilitates efficient interchangeability between the modules, while also allowing each module to be used separately.

The open-source, integrated planner introduced in this chapter can be used for autonomous navigation of mobile robots in general, including autonomous driving applications. It is designed to use vector maps or road network maps and all of the discrete information they contain, such as the locations of traffic lights, traffic signs, intersections, stop lines, and so on, which is one of its main advantages over other

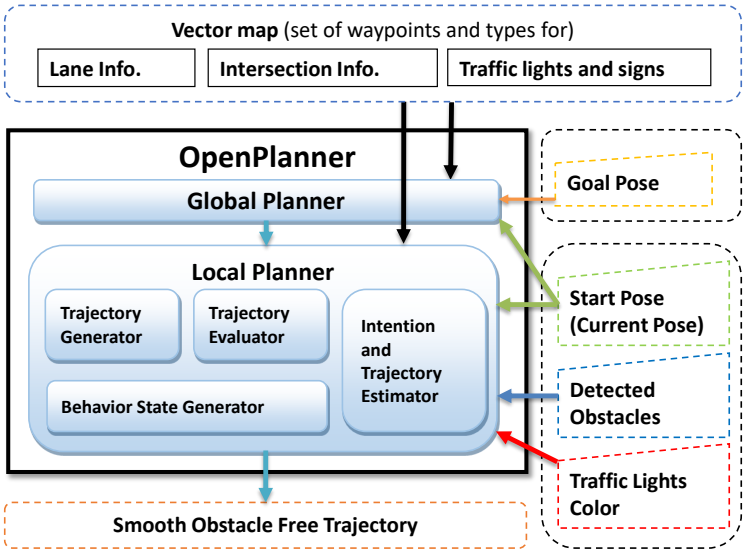


Figure 3-1: The Integrated Planner Architecture. It is developed as separate modules with easy to use well defined module interface. which makes it efficient and easy to use.

open-source, general purpose planners like OMPL and Navigation Stack. Use of vector maps allows easier and faster global and local planning by removing kinematic optimization from the planning equation. Instead, vector maps handle that problem. Of course, global planners like RRT* [61] and Hybrid A* are useful in undefined spaces, such as parking lots and off-road areas. In these situations, we can switch to a free space planner for global planning and still use OpenPlanner's behavior state machine and local planner. Another advantage our planner has over Navigation Stack is that it can be used with non-holonomic platforms.

The function of behavior state machines is hard to generalize because the manner in which they are used differs from one robot to the next. ROS, for example, provides basic behavior state machine functionality which the user can customize. OpenPlanner also provides basic behavior state machine functionality, and adding new states is as easy as with ROS. OMPL, on the other hand, doesn't provide a state machine or discrete behavior planning. Regarding the mapping requirements of these planners, both Navigation Stack and OMPL require cost maps, which OpenPlanner does not

require, unless switching to a free space planner; only a vector map is needed. In summary, OpenPlanner is more suitable for autonomous robot navigation systems that obey traffic rules, since it requires only a vector map and a goal location for global planning, and only requires current position and detected obstacles for local planning and behavior state generation. Recently, a complete autonomous driving planner has been release as part of the Apollo Open-source project [59]. Like OpenPlanner, Apollo's planner has a global planner, local planner and behavior planner. But currently, OpenPlanner edges out Apollo's planner on several key points:

- OpenPlanner can rely exclusively on road network maps to navigate, so roads do not need to be driven beforehand.
- OpenPlanner supports standard and open-source map formats, such as OpenDRIVE [60] and Lanelet2 [62].
- OpenPlanner supports multiple platforms and environments.

Figure 3-2 shows a summary of problems with existing open-source planners, and how our integrated planner resolves these issues.

Challenges with other open-source planners	How OpenPlanner overcomes these challenges **(Contributions)**
Difficult to use their internal functions separately.	Includes several integration interfaces, such as Library APIs, and ROS nodes. Easy to use, with several tutorials included.
Designed to support specific environments (indoor, outdoor, highway, etc.).	Environment support depends on map availability.
Designed for one target platform (differential robots, slow speed mobility, autonomous vehicles, etc.).	Can support multiple platforms by tuning only a small number of parameters. Tested with different platforms (mobility scooter, full size autonomous vehicle, robot boat).
Map support: Either cost maps are used, which are difficult to create and maintain, or non-standard maps are used, as in the case of the Apollo planner.	Uses standard opens-ource road network maps (vector maps, OpenDRIVE, lanelet2 and custom KML maps). Mapping tools are also available.
Missing functionality: Currently only the Apollo planner includes all planning modules (Global, Local, Behavior).	All planning modules are provided separately as ROS nodes.
Simple object representation is used to improve performance, such as bounding boxes.	Precise object representation to improve local planning accuracy.
Performance is not guaranteed.	Even with 100 obstacles, performance of the entire planning stack is still more than 10Hz.

Figure 3-2: Contributions of the proposed OpenPlanner integrated planner, in relation to issues with current, state-of-the-art, open-source planners.

As a result of the contributions shown in Figure 3-2, OpenPlanner is being used in several international projects:

- **Roboat project**: The objective of this project to encourage the use of Amsterdam's canals for transportation, using a fleet of autonomous boats [22][23].

- **Autonomous driving map editor**: An open-source map editor for autonomous driving applications, created at the University of Konkuk, Korea [24].

- **ADAS Demo**: Advanced Driver Assistant System (ADAS) demo for a major Hong Kong based technology company.

In addition, the paper in which our integrated planner was introduced has been downloaded over 3,000 times, and cited 17 times.

3.2 Global Planning

Autonomous vehicle planning is divided into two main categories, depending on the driving environment. The first type of planning involves unstructured environments like those encountered during off-road driving or in parking situations, locations in which we cannot use vector maps. The most suitable type of mapping in these situations is a cost map. The second type of environment involves structured environments where we have clearly defined roads, traffic lanes, intersections, etc., as well as traffic signs, all of which can be described in vector maps.

The global planner handles path routing. It takes the vector map, a start position and a goal position as input and then finds the shortest or lowest cost path using dynamic programming [16]. The global planner used by OpenPlanner can support complicated vector maps, but for this study the maps used were very simple. The entire map for the Tsukuba RWRC is shown in Figure 3-3. We annotated the map manually with traffic rules and features, such as traffic lights and stop lines, from start to goal.

Figure 3-3: Tsukuba Real World Robot Challenge vector map.

3.2.1 Road Network Maps Utilization

One of the most widely used approaches for autonomous vehicle navigation is the use of vector maps, sometimes called high definition road network maps to differentiate them from maps used in geographic information system (GIS) applications such as open street maps [29]. Vector maps include data that autonomous navigation modules need to make sense of the surrounding environment. Table 3.1 shows some common components of vector maps and the potential uses of each component in autonomous driving systems. OpenPlanner uses a 2.5D map, which means it includes elevation information used only when needed. This allows increased planning performance since most planning is done in 2D, but 3D information can also be used in rare situations, such as when very steep slopes are encountered.

Representing the center of lanes in vector maps with high order polynomials will help us interpolate way-points having the required density, but the disadvantage of using polynomial curves is computational overhead. There is no problem if the map is loaded from a file once, but if the map is updated from a map server it can slow the planning process. For this reason, we developed an efficient algorithm to adjust the

Table 3.1: Vector map components, in order of importance to most planning algorithms.

Component	Potential usage
Lanes network: Lane ID, Previous lanes, Next lanes, List of central way-points	Generating global reference path and lane change commands using global planner
Traffic light	Traffic light detection
Stop lines	Stop lines for traffic lights, stop signs and intersection)
Traffic signs	Signs detection and planning
Lane boundaries and road signs	Safety and localization

density of lane center lines, as shown in Table B.1 in appendix B. The center lines of manually created maps are not smooth due to human error, so an additional step of smoothing using the conjugate gradient (CG) method [63] is applied.

Although the Tsukuba RWRC map is very simple, Figure 3-3, as is our testing map for the Nagoya University campus Figure 3-4, OpenPlanner also supports road network compatible vector maps such as the one shown in Figure 3-5.

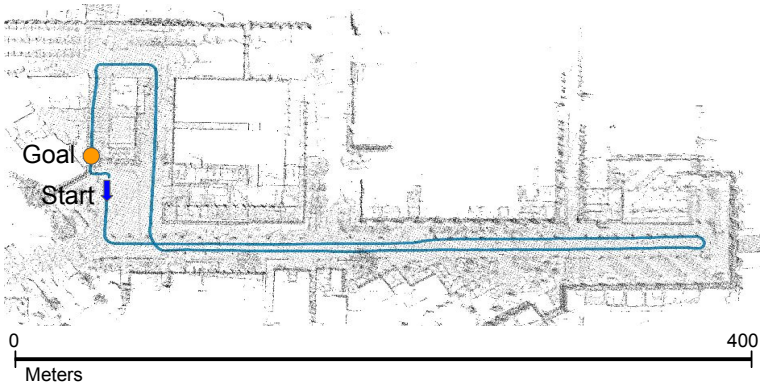


Figure 3-4: Experimental vector map for Nagoya University.

3.2.2 Global Path Cost Function

The main objective of path planning is to find the optimal path from a starting point to a goal, but in structured environments we must follow the traffic rules, such as driving in the center of the lane, traveling in the right direction, changing lanes only when allowed to and moving into the correct lane to make right or left turns.

When using dynamic programming to find the optimal path, we trace possible

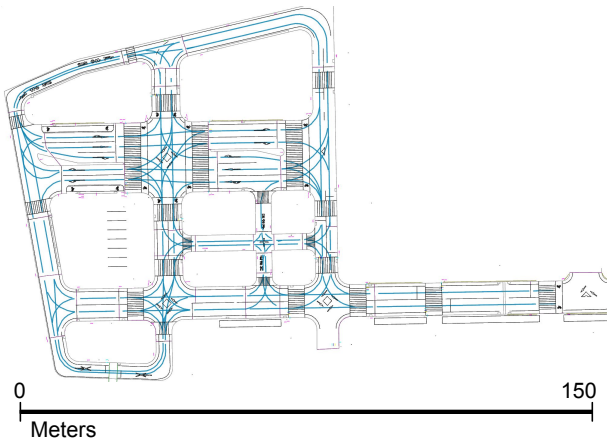
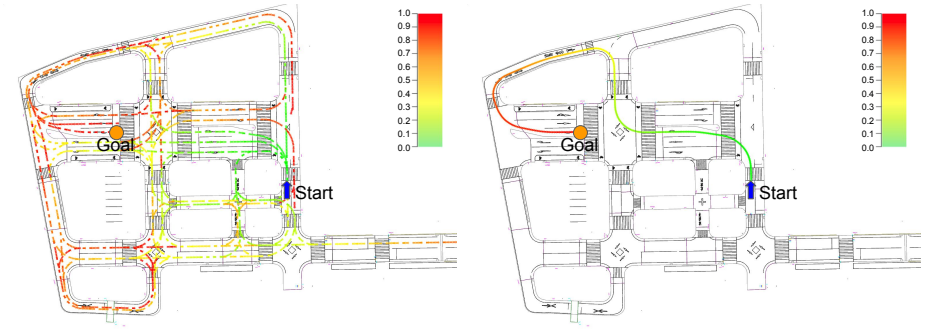


Figure 3-5: Vector map with complex structures.



(a) (b)

Figure 3-6: Searching the map for the shortest path. Line color indicates route cost. Cost is represented by distance, green color is closest distance to Start and red color is max distance reached at the Goal position.

routes forward starting from the current position of the vehicle to the goal. During route tracing, we construct a tree of possible paths which follow the defined rules until the vehicle reaches the goal, as shown in Figure 3-6 (a). The colors code in Figure 3-6 indicate relative distance from Start position, green is closest to the start position and red is furthest point reached (Goal). Once we reach the goal, we trace the route back from the goal to the start position, annotating the path with all the information the local planner needs to generate a local trajectory, as shown in Figure 3-6 (b). The local planner needs to know traffic direction, lane change locations, positions of stop lines, positions of traffic lights and speed limits.

Sample global paths generated for a complex vector map are shown in Figures

3-7 and 3-8, the latter of which includes a lane change. Color represents relative distance from start to goal. In Figure 3-4, we show the global path for one of the testing environments, which simply represents the same vector map shown in Figure 3-4. Tables B.2 and B.3 in appendix B show the algorithms used to find the global path.

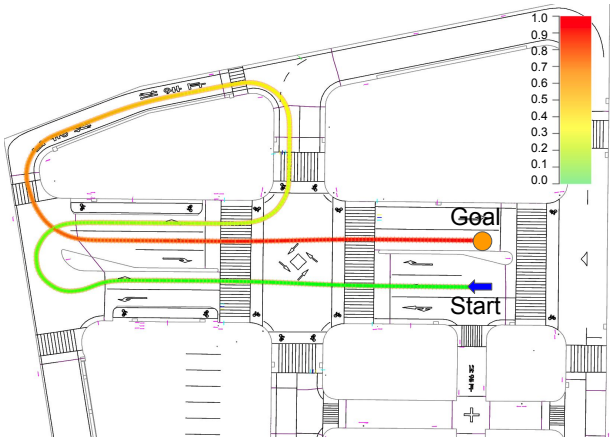


Figure 3-7: Example of complex global planning, disabling lane change. Taking maximum rout available. Color represents relative distance, green color is closest distance to Start and red color is max distance reached at the Goal position.

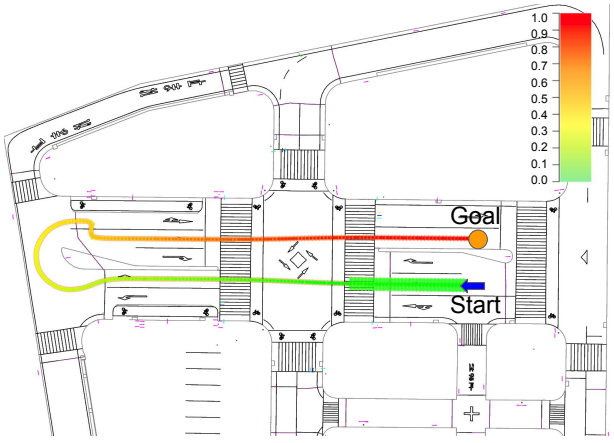


Figure 3-8: Global planning including lane change. Taking shortest rout available. Color represents relative distance, green color is closest distance to Start and red color is max distance reached at the Goal position.

3.3 Trajectory Planning

A local trajectory planner is a set of functionality that generate a smooth trajectory which can be tracked by path-following algorithms, such as Pure Pursuit [64]. For OpenPlanner, we adapted the roll-out generation approach (Fig. 3-9), in which the behavior generator can demand a re-plan at any time to generate fresh, smooth, roll-out trajectories. Re-planning will be discussed in detail in the next subsection.

Inputs for the local path planner are the global reference path and the current position. Several candidate trajectories are then generated as roll-outs and the local planner selects the one with the lowest collective cost. Figure 3-9 shows seven possible rollout trajectories, including the center one. We used a modified version of the Stanford approach presented in [16].

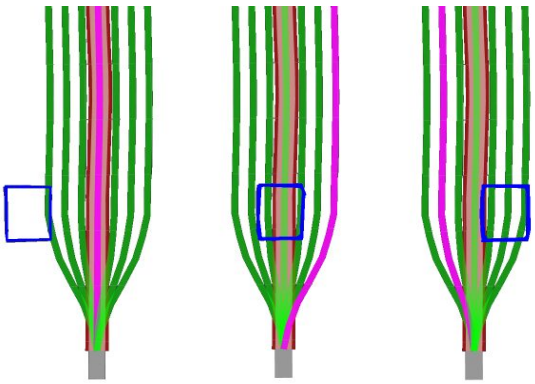


Figure 3-9: Local Planner in action, in a) central trajectory is free, in b) obstacle blocks central trajectory so the most feasible one was the most right trajectory, in c) the most feasible one was the second on the left.

3.3.1 Trajectory Generation

Roll-out generation needs to be executed in real time, as it is a basic requirement that all local planners be able to work in real time. The target processing time therefore is a maximum of 0.1 seconds so that the controller can respond quickly to changes in velocity. The inputs of the roll-out generation algorithm are current position, planning distance, number of roll-outs and the next section of the global path. The

output is n smooth trajectories, running from the center of the vehicle out to the maximum planning distance.

The sampled roll-outs are divided into three sections as shown in Figure 3-10. The closest section to the vehicle is the car tip margin, which is the distance from the center of the robot to the point of lateral sampling, the length of which determines the smoothness of steering when switching between trajectories. The next section is called the roll-in margin, which is the distance from the outer limit of the car tip margin to the point of parallel lateral sampling, the length of which is proportional to the vehicle's velocity. The faster the vehicle is traveling, the longer this section should be to generate smooth change. The section farthest from the vehicle is called the roll-out section, which runs from the outer limit of the roll-in zone to the end of the length of the local trajectory. Straight-forward lateral sampling is performed by moving perpendicularly from the global path for a fixed distance, which is called the roll-out density.

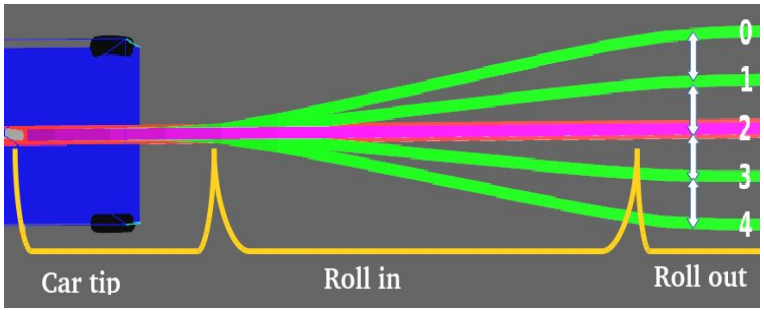


Figure 3-10: Sections for generating roll outs.

The generation of roll-outs by the local trajectory planner algorithm includes three main steps, the first of which is extracting the section of interest from the global path using the current position of the vehicle and maximum planning distance. The second step is to sample the new perpendicular way-points which correspond to the extracted section of the global path. The sampling starts from the car-tip margin with a lateral distance of zero, then increases gradually to reach the roll-out density calculated using each trajectory index at the end of the roll-in margin. The third step is to smooth each sampled trajectory using a conjugate gradient, which is non-linear iterative optimization technique that eliminates the discontinuity of roll-

outs resulting from the sampling step. This also improves curvature, which leads to smoother steering.

The density of trajectory vertices (way-points) is adjusted using piece wise interpolation, as shown in Table B.1 in appendix B. Many parametric interpolation techniques are very sensitive to input noise and propagate that to the output (e.g., cubic splines can lead to arbitrarily large oscillations in the output as input vertices get closer to each other) [65]. Therefore, we use a combination of piece wise interpolation and conjugate gradient smoothing to generate smoother trajectories. The resulting trajectories are generally kinodynamically feasible because we are using vector maps, thus we assume that all lanes are kinodynamically feasible. Figure 3-11 shows the steps of roll-out generation, and implementation is demonstrated in Table B.4 in appendix B.

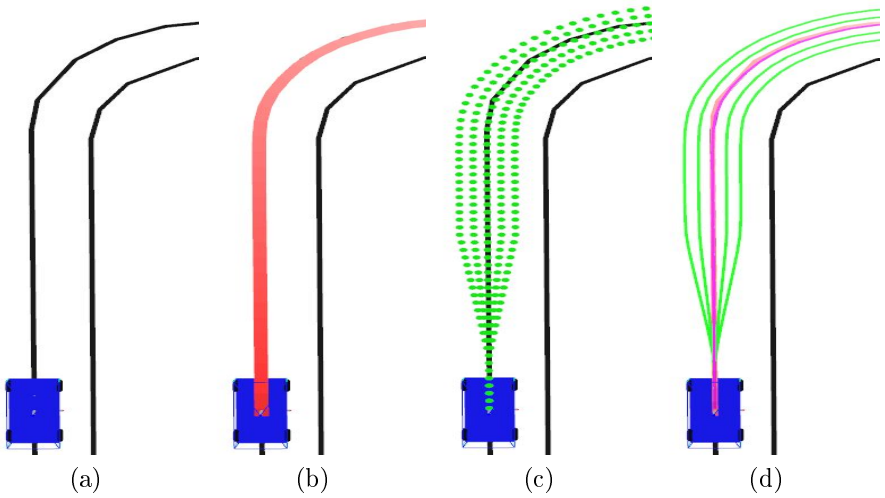


Figure 3-11: Steps for generating local trajectories: (a) original map, (b) path section extracted from global path, (c) sampling, (d) smoothing using conjugate gradient.

3.3.2 Trajectory Selection Cost Function

In addition to roll-out generation, the other important function of the local planner is obstacle avoidance within a lane, i.e., swerving. Obstacle avoidance is the process of selecting the best possible trajectory from the roll-outs generated using algorithm in Table B.4 in appendix B. Inputs to the obstacle avoidance process are the roll-outs

and detected obstacles, and the output is the selected trajectory. We use an additive cost function to evaluate each trajectory, which calculates three different normalized cost measurements, priority cost, collision cost and transition cost, the smallest of which is selected.

Obstacle detection is achieved using another module in Autoware [18] which outputs two types of obstacle representations, bounding boxes and clusters of point cloud data, as shown in Figure 3-12. Obstacle representation is essential for both accuracy and performance, and by using bounding boxes we can dramatically improve obstacle detection performance, but at the expense of accuracy. Using the point cloud data greatly improves detection accuracy but degrades performance drastically. We solved this trade-off problem by using only a sample of the contour points from the clusters of point cloud data, with a maximum of 16 points for each obstacle.

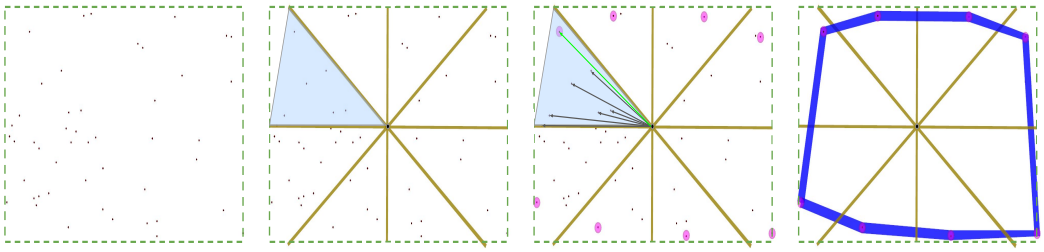


Figure 3-12: Obstacle data representation using only the contour points from the point cloud data. In (a) we show sample random point cloud points. Step 1 of the contour calculation is shown in (b); we find the point cloud center point and from that point we divide the point cloud into 8 quarters, number of quarters is a parameter and could be changed to get higher resolution contours. In (c) we show step 2 of the process, for each quarter we calculate Euclidean distance between center and each point belongs in this quarter, then we find the point with maximum distance. (d) shows the final contour result with only the selected points from each quarter.

The maximum number of contour points is one of the parameters of the local planner, and by increasing this number we can achieve finer representation, which leads to more accurate obstacle avoidance. Figure 3-12 (b) shows an example of obstacle detection using 16 contour points. Contour representation is calculated in three stages: first, we divide the xy plane into n sectors; second, we find the distance

and angle between each point and the center point, and use the angle to assign the point to a sector; third, we select the final contour points, which will be the points at the maximum distance from the center of each sector.

Center cost

Center cost constrains the vehicle to drive along the center of a lane at all times, and each roll-out is calculated using the absolute distance from this lane-centered trajectory.

Transition cost

Transition cost constrains the vehicle from jumping roll-outs, which contributes to smoother swerving. This cost is calculated using the normalized perpendicular distance between roll-outs as well as the currently selected trajectory.

Collision cost

Collision cost is calculated in two stages to improve performance. In the first stage, we test each trajectory by measuring the distance from the obstacle contour points to each generated trajectory. Since all of the trajectories are parallel to the center trajectory after the roll-in section, we don't have to apply an explicit test after the roll-in distance. The obstacle test is achieved using a "point inside a circle" test, where each contour edge provides the test points, circles centers are the way-points, and the radius of each circle is half the vehicle width plus a detection margin of error.

The second stage of collision cost calculation is checking distances between the detected obstacles and the generated trajectories after the roll-in limit. After the roll-in limit all generated trajectories are parallel, so we don't need to calculate the collision cost for each trajectory separately. We calculate the distance from the contour points of the detected obstacles to the central trajectory, then use the signed distance from each trajectory to the central trajectory to find the collision cost for each trajectory. Figure 3-13 shows color coded center costs, and Figure 3-14 illustrates the normalized total cost when there is an obstacle.

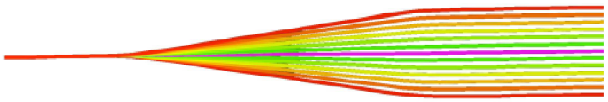


Figure 3-13: Center cost keeps the robot in the center of a lane. Selected trajectory color is pink, with other trajectories colors represent cost, with green is the smallest cost and red is the largest.

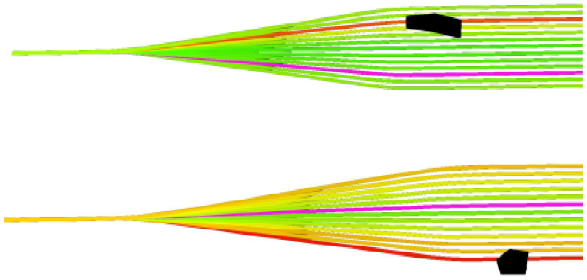


Figure 3-14: Effect of an obstacle on cost calculation.

3.4 Behavior Planning

The behavior state generation module of OpenPlanner functions as the decision maker of the system. It is a finite state machine in which each state represents a traffic situation. Transitions between states are controlled by intermediate parameters calculated using current traffic information and pre-programmed traffic rules. Figure 3 shows the currently available states in the OpenPlanner system.

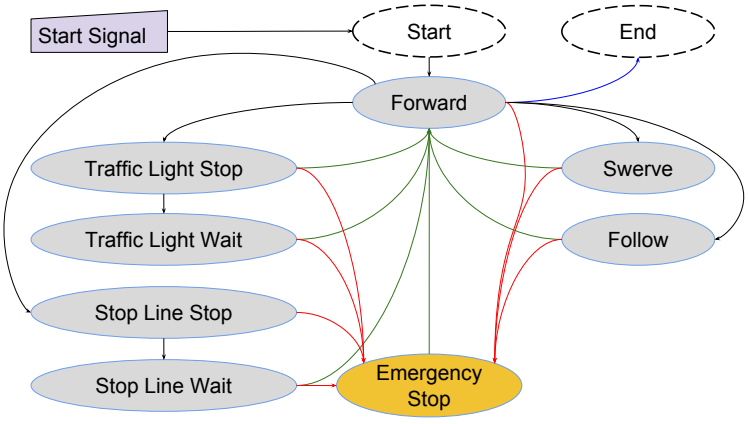


Figure 3-15: Current System Behavior States.

Situations like stopping at a traffic light, deciding to change lanes, stopping and waiting at a stop sign and yielding to pedestrians are difficult to handle using one algorithm. Like other traffic rules, these events are definite in nature but the rules vary from country to country. Moreover, special traffic rules or objectives could be added or disabled any time. We call responses to these events behaviors, tasks, objectives, states or situations. In this study, we use the term "behavior state" to represent all of these event responses, as well as to refer to the transition rules between these states. Figure 3-15 shows the behavior states used for OpenPlanner during the Tsukuba Real World Robot Challenge. Table 3.2 shows the transition rules for each state.

Table 3.2: Behavior states transition conditions.

State	**Switch to**	**Condition**
Start	Forward	receive start signal from joystick
Forward	Swerve	current trajectory is blocked, not all trajectories are blocked
Forward	Follow	all trajectories are blocked
Forward	Traffic light stop	traffic light is red within the stopping distance range
Forward	Stop sign stop	stop sign within the stop distance range
Forward	Mission accomplished	goal position within the distance range
Swerve	Follow	all trajectories are blocked
Swerve	Forward	drive parallel to the center
Follow	Forward	not all trajectories are blocked
Traffic light stop	Traffic light wait	not green light and velocity is zero
Traffic light stop	Forward	green traffic light
Traffic light wait	Forward	green traffic light
Stop sign stop	Stop sign wait	velocity is zero
Stop sign wait	Forward	after stopping for time range
Any state	Emergency stop	receive emergency stop signal
Emergency stop	Forward	no emergency stop signal

There are several parameters controlling transitions between states. These parameters are calculated deterministically every iteration. Theoretically, a probabilistic approach should result in smoother transitions, but it is slower and more complicated to implement and maintain over a wide range of applications. One solution to this problem is to introduce timers and counters. For example, when an obstacle is moving very close to a threshold, the behavior generator will rapidly switch back and forth between the Swerve and Follow states. A counter or timer can break this cycle.

Another situation in which counters will give better results is when a traffic light switches to red or green and the light detector is not reliable enough to handle the change. In such cases, it is necessary to receive the signal multiple times to assure the reliability of the signal and switch to the next state. Therefore in the initialization of each behavior state we set a minimum transition time, so that a state will keep executing itself unless a set amount of time has elapsed or emergency state conditions are met.

3.5 Experiment Results

In this study we use an Ackerman-based steering robot, based on a mobility scooter, which was used in the Tsukuba RealWorld Robot Challenge (RWRC) [66] where we tested the planner. RWRC is an annual mobile robot challenge held in the city of Tsukuba, Japan. The robots participating in the event must be able to achieve accurate localization and autonomous navigation in a dynamic environment, handle traffic lights and street crossing situations, navigate through an automatic sliding door, go inside a shopping mall and search for a designated person. Our goal in participating in the RWRC was to use OpenPlanner to achieve as many of these tasks as possible. Many innovative and effective planning algorithms are developed for this challenge every year, but unfortunately most of these planners are proprietary. Every year, new participants develop their own planning systems from scratch, and only a limited number of the outlines and details of these systems are described, usually quite briefly, in published papers. Thus, one of the motivations for us to develop an open source planner was to provide the robotics community with a planner that is easy to understand and use which can also be continuously developed by its users.

The first field test was conducted on the campus of Nagoya University, and a diagram of the route is shown in Figure 3-16 . The second field test took place at the Tsukuba RWRC event, the vector map of which is shown in Figure 3-3 . The objectives of these experiments were to test global planning performance from any current position to any goal position on the map and to evaluate performance of

obstacle avoidance, stops at stop signs and stops at traffic lights. Additionally, we also evaluated smoothness, performance, accuracy, practicality and usability. In the simulation environment, we used real life vector maps composed of lines, intersections, stop lines and traffic signs as shown in Figures 3-5 and 3-6. For field experiments, we added traffic information to the map manually.

Figure 3-16: Experimental vector map with stop lines and traffic light.

The experiment platform is shown in Figure 3-17, same platform is used in [67]. It is a modified mobility scooter so it could be controlled by computer. It includes one HDL32 Velodyne LIDAR sensor which is used for localization and object detection. In addition to the 3D LIDAR we use three 2D LIDAR for curbs and near obstacles detection. For the software part we had multiple ROS nodes for localization, obstacle detection, control, global planning, local planning and path following. In appendix A we provide technical information about OpenPlanner for ROS users.

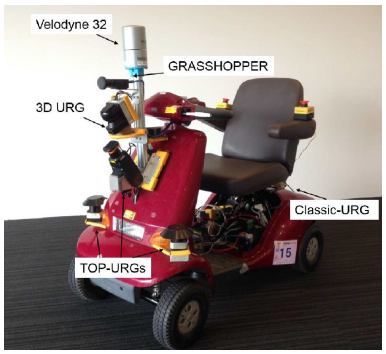


Figure 3-17: Tsukuba Challenge hardware platform.

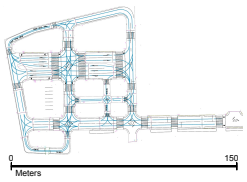
In this section we will discuss the qualitative results first, then present key results

Table 3.3: Parameters configurations.

Parameter	Configuration
path density	Usually keep path density between 0.25 and 1 meter, with 0.5 is most recommended. Sure for very small robots higher path density is required but we never tested the planner on such platform
roll out numbers	The more roll out we have we achieve smoother avoidance but slower performance. We used from 6 to 12 roll outs with 8 giving the best combination of accuracy and smoothness
sampling tip margin	Length of the vehicle gave us good results which was 1.2 meters
sampling out margin	Affects smoothness off obstacle avoidance, but setting it too big will delay the avoidance until the vehicle become very close to the obstacle, good values for it was between 5 and 8
following distance	This value should be greater than "distance to avoid" and good values for "following distance" is 12 meters
distance to avoid	Good value for this parameter is 8 meters

from our simulation, experiments on the Nagoya University campus and participation in the Tsukuba RWRC event. Both the simulation and field tests had positive results. The table 3.3 shows the used parameter configurations for simulation and field experiments. Summary of our experiments is listed in table 3.4.

Table 3.4: Experiments.

Experiment	Environment	Objectives
Simulation		**- Stability and correctness:** the planner should correctly plans trajectory and behavior every time. **- Complex routing:** the global planner can find shortest path in a complex map. **- Behaviors coverage:** the planner can generate all the designed planning behaviors.
Nagoya University		**- Performance:** the planner can perform real time at minimum 10hz. **- Parameter tuning:** Smooth actions by tuning the parameters. **- Accuracy:** the planner was able avoid obstacle accurately even in narrow passages. **- Completeness:** the planner could handle different traffic situations.
Tsukuba Challenge		**- Dynamic routing:** automatic rerouting when perception based behavior is not available. **- Highly dynamic:** the planner should navigated through busy unexpected environment.

3.5.1 Qualitative results

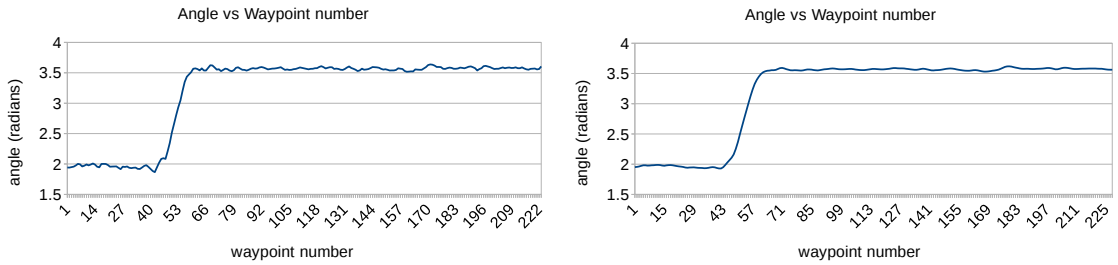
The first qualitative aspect of our experiment is to evaluate stability, which means OpenPlanner should work all the time. Even if faulty data is provided, meaningful error messages should appear but the planner should never stop operating or crash.

We tested this in a simulated environment by running the planner from any start point on the map, stopping localization at some point, and by jumping from point to point on the map manually and then switching to autonomous mode.

The second qualitative aspect is completeness, which means the system delivers smooth switching between different states and never remains stuck in one state. Experiments showed that the local planner stops successfully at stop signs and at traffic lights when they are red. Switching between follow, forward and obstacle avoidance states also worked properly, but smoothness of transitions depended on the reliability of the obstacle detection node. When encountering an obstacle-free path while navigating the map in Figure 3-16, the resulting sequence of driving behaviors were: Forward, Stop Sign, Wait Sign, Forward, Light Stop, Wait Light, Forward, Finish.

3.5.2 University campus experiment

Here we will concentrate on one section of the campus field test map, shown in Figure 3-16, which consists of two straight lines, one curve, one stop line and one traffic light. Figure 3-18 (a) shows the curvature of the generated path, and the smoothed output of that path is shown in Figure 3-18 (b). Smoothing improves path following performance without the need to relax the control parameters.



(a) (b)

Figure 3-18: The angle of each way-point shows curvature of the generated path with and without smoothing.

Results when navigating the map section shown in Figure 3-16 are displayed in Figure 3-20, which shows the simulation results when navigating this section without any obstacles. Behaviors are represented by the blue line and behavior ID values. The

orange line represents the trajectory index, which is the currently selected roll-out number. In this experiment we used 7 roll-outs, with 3 representing the number of the center trajectory. Figure 3-19 illustrates the behavior transition flow during this experiment.

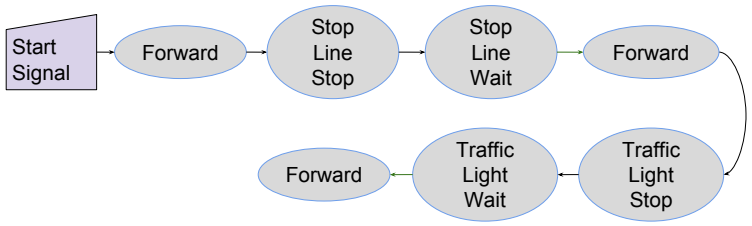


Figure 3-19: Behavior state flow while navigating the simulated vector map without obstacles.

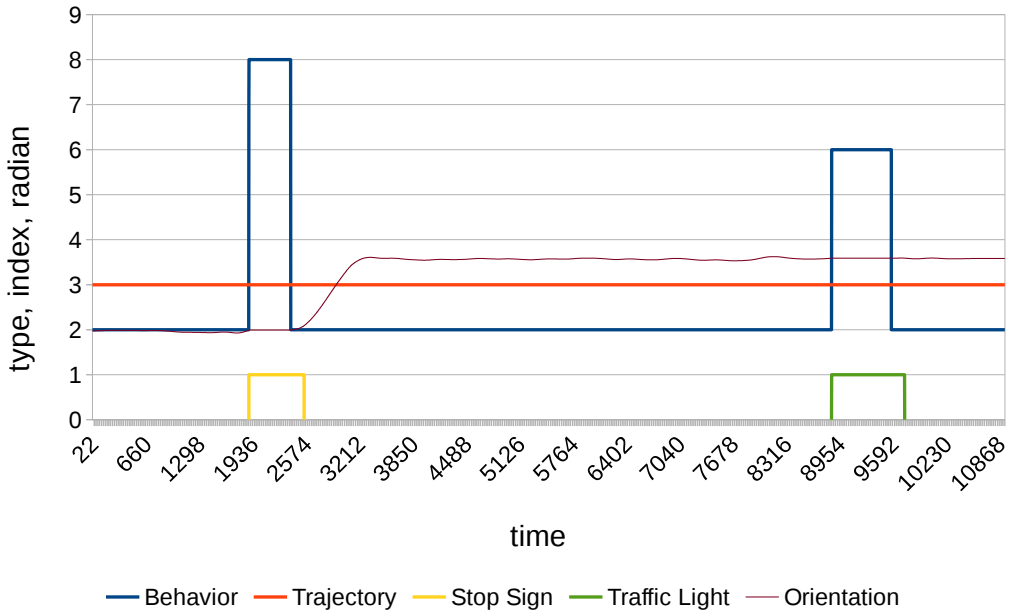


Figure 3-20: Simulation test results when navigating without obstacles. Behavior ID values: 2 = move forward; 5 = stop for traffic light; 6 = wait for traffic light; 7 = stop for stop sign; 8 = wait for stop sign; 11 = avoid obstacle.

Using the OpenPlanner in simulation mode enabled us to insert obstacles of random sizes. We repeated the previous experiment after adding two obstacles while the robot was moving, one obstacle before the stop sign and the other after the stop sign. Figure 3-21 shows that both state transition and trajectory switching results

were smooth. Figure 3-22 is a diagram of the behavior state transition flow when navigating this section with obstacles.

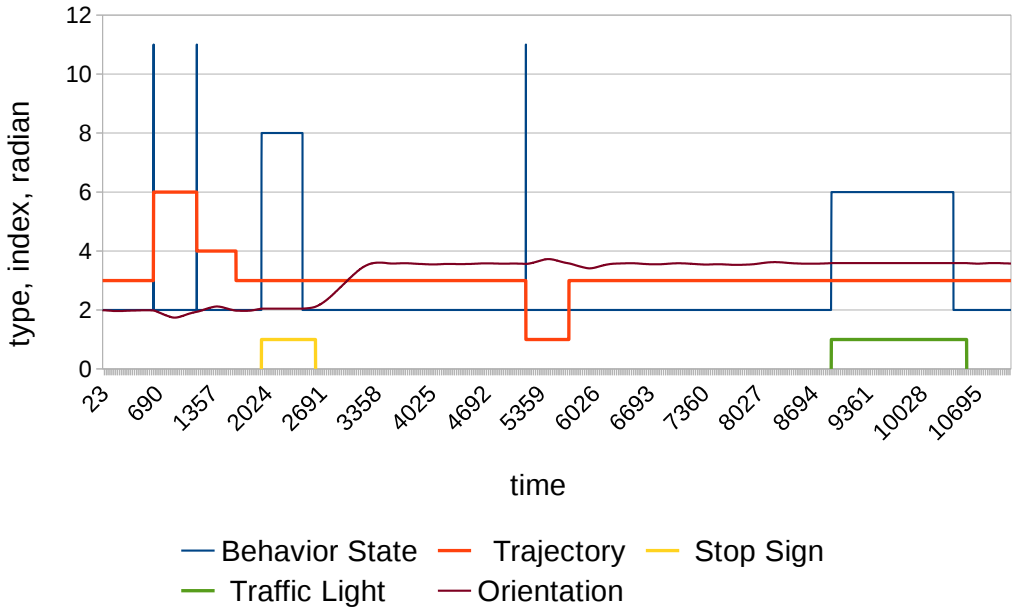


Figure 3-21: Simulation test results when navigating with obstacles. Behavior ID values: 2 = move forward; 5 = stop for traffic light; 6 = wait for traffic light; 7 = stop for stop sign; 8 = wait for stop sign; 11 = avoid obstacle.

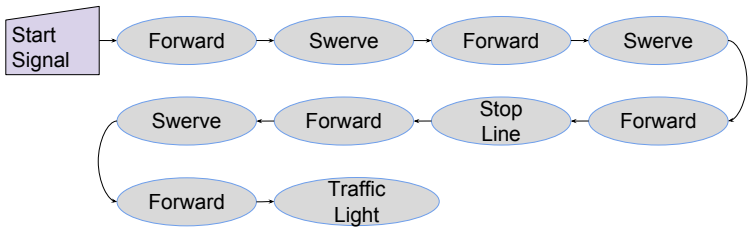


Figure 3-22: Behavior state flow while navigating the simulated vector map with obstacles.

Field experiment results for the same map section, shown in Figure 3-23, were similar and generally stable, with the vehicle stopping at the stop line and traffic light and avoiding obstacles when necessary. However, noisy input data resulted in an additional state transition.

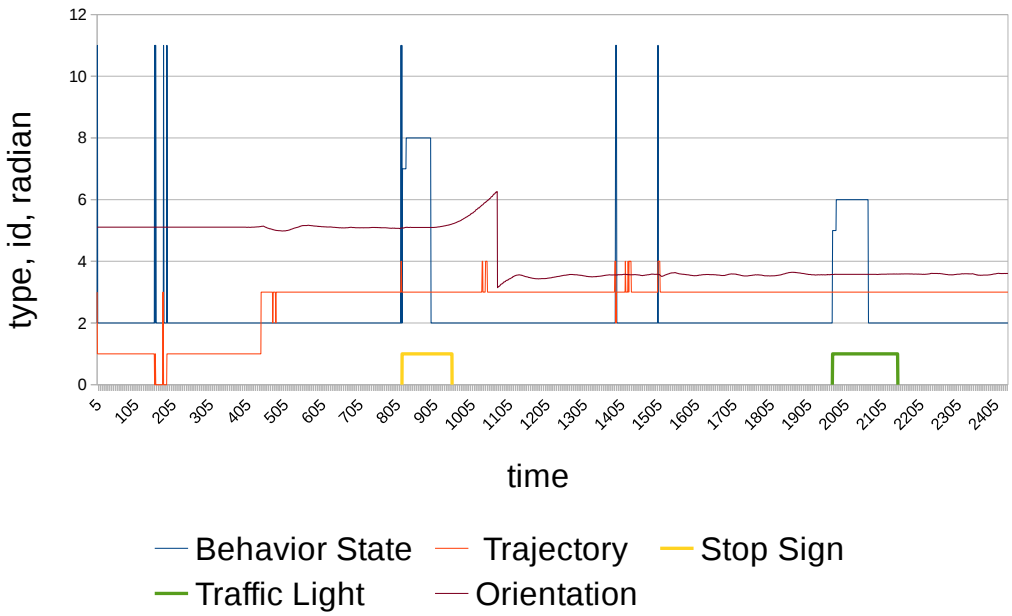
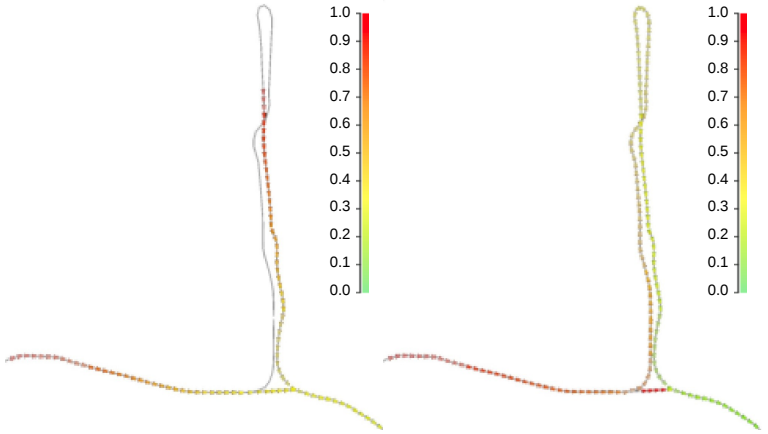


Figure 3-23: Field test results. Behavior ID values: 2 = move forward; 5 = stop for traffic light; 6 = wait for traffic light; 7 = stop for stop sign; 8 = wait for stop sign; 11 = avoid obstacle.

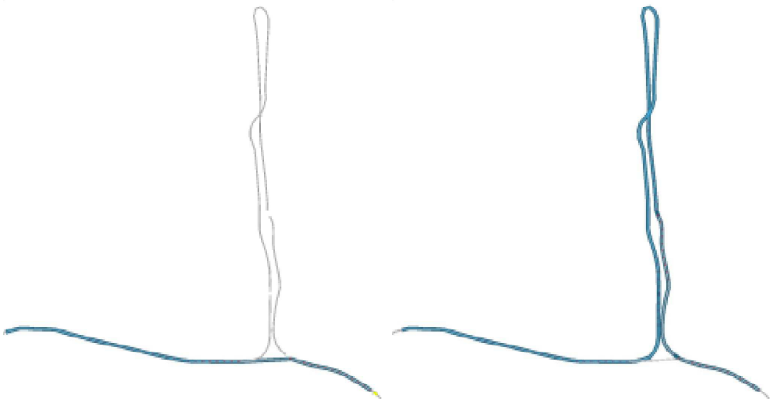
3.5.3 Tsukuba RWRC experiment

One of the most difficult tasks during the Tsukuba event was the street crossing, which required stopping at a traffic light, crossing the street, making a very tight U-turn, stopping at a second traffic light and crossing the street again. By using vector maps we were able to dynamically choose to continue through this stage or bypass it. The default behavior of the planner is to find the shortest route from the current position to the goal, so normal route selection would result in the route shown in Figure 3-24 (a). If we wanted to attempt the task and cross the street, we could simply increase the cost associated with the shortcut. Figure 3-24 (b) shows the planner choosing the longer route to avoid the high cost assigned to the shortcut. The resulting global paths are shown in Figures 3-25 (a) and (b) respectively. Behavior states when following both routes are simulated in Figure 3-26 (a) and (b).



(a) (b)

Figure 3-24: Section from Tsukuba challenge path represented by red rectangle in Figure 3-3. Global planning dynamic cost calculation, (a) without and (b) with a high shortcut penalty.

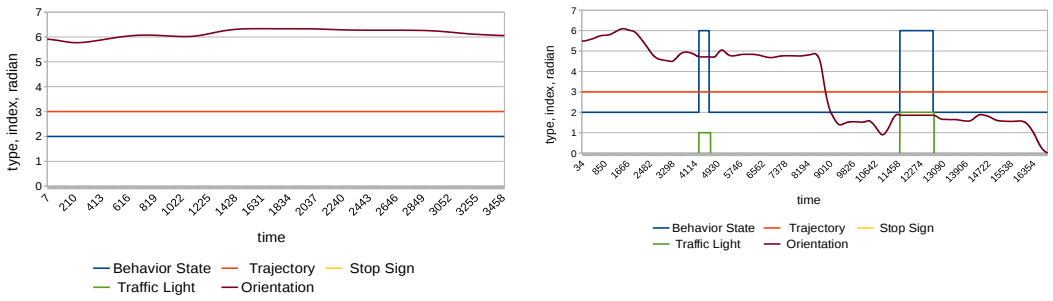


(a) (b)

Figure 3-25: Section from Tsukuba challenge path represented by red rectangle in Figure 3-3. Global planning final paths, (a) without and (b) with a high shortcut penalty.

3.5.4 Performance

In the campus field experiment, we were able to achieve real time performance of 14.6 iterations per second for local planning and behavior state generation, while detecting an average of 62 obstacles and an average total of 1,255 contour points (Figure 3-27). For each iteration we calculate the obstacles contours, track the obstacles using a Kalman filter, calculate the costs and then generated an new roll-outs if needed.



(a) (b)

Figure 3-26: Behavior state results when selecting the shortest route (top) or choosing to cross the street (bottom). Behavior ID values: 2 = move forward; 5 = stop for traffic light; 6 = wait for traffic light; 7 = stop for stop sign; 8 = wait for stop sign; 11 = avoid obstacle.

In Figure 3-27 yellow curve represents the execution time for cost calculation step and blue curve represents the number of detected obstacles, we can see that both curves almost have the same trend; when number of detected obstacles increases the execution time increases and vice versa. Cost calculation is clearly a performance bottleneck. At this point, no performance enhancement techniques were used, not even a compiler optimization directive, but we were still able to achieve real time performance. The next objectives for OpenPlanner are smoother obstacle presentation and a larger number of generated roll-outs.

3.5.5 Accuracy

Our mobility scooter was able to avoid static and moving obstacles successfully, depending on localization accuracy. Our testing platform had an original localization accuracy of within 10 cm, so we added 20 cm to its width and length as a safety margin for the robot As a result, in some cases the robot got as close as 5 cm to an obstacle while avoiding it. Recorded videos of experiments including visualization of environment is shown at OpenPlanner channel [68] on youtube. We tested several parameters using the same route section illustrated in Figure 3-28. Since we added a 20 cm safety margin to the width and length of the robot, it became difficult for the robot to drive in a straight line in the narrower sections. By changing the car tip and

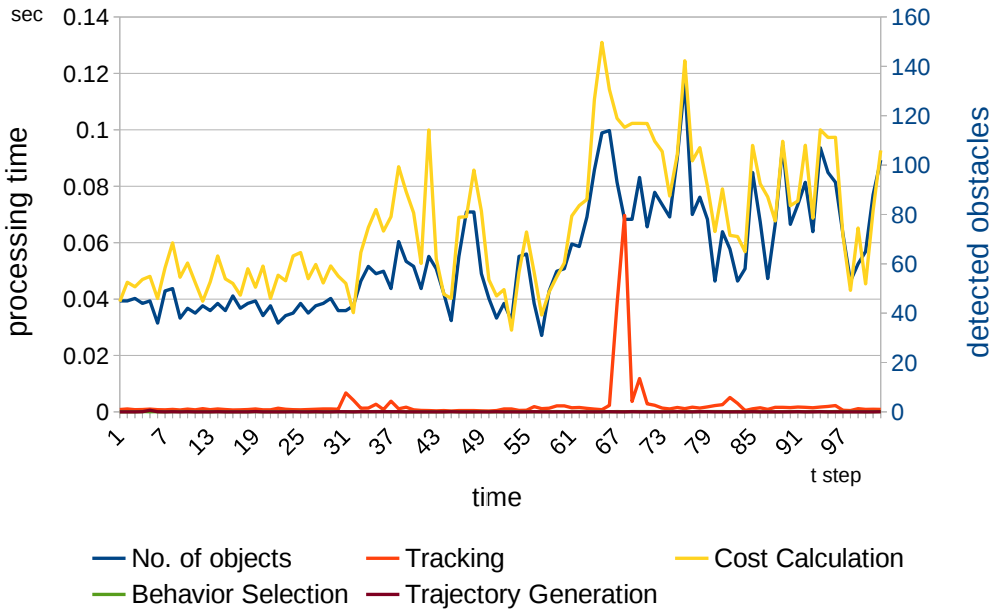


Figure 3-27: Performance results of campus field test. Y axis on the left shows processing time used for calculating (obstacle tracking, trajectory cost, behavior selection, and trajectory generation) in seconds, with relation to number of detected obstacles represented by right Y axis.

roll-in parameters, we were able to achieve a smoother driving path when traveling through this section. Figure 3-29 shows data from several trials while navigating through the section shown in Figure 3-28.

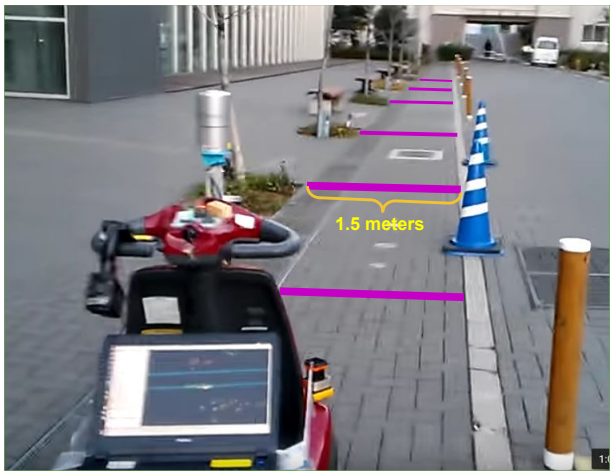
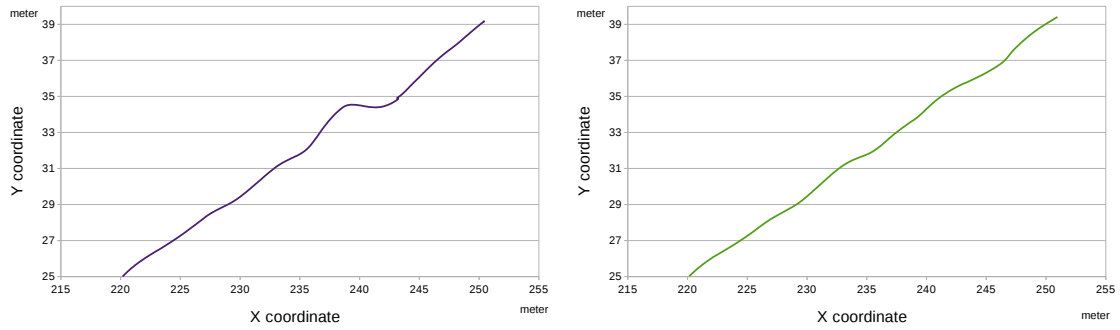


Figure 3-28: A narrow pathway was used in the campus testing area to evaluate performance when avoiding nearby obstacles.



(a) (b)

Figure 3-29: Driving between cones and trees generated many changes of direction. In (a), using smallest roll-in margin, the motorized scooter changes direction more quickly but not smoothly. In (b) we used bigger car tip and roll-in margins, so the results became smoother.

3.6 Conclusion and Summary

OpenPlanner is an integrated planner in the sense that it performs global, local and behavior planning. It is an open source planner for the robotics community to take advantage of, use, modify and build on. In this chapter we explained our methodology, vector map design and the dynamic programming of the global planner. Also we explained how the local planner generates trajectories and safe trajectory selection criterion. We outlined behavior state generation using a state machine transition matrix and explained the functions and usage of related ROS nodes. We conducted a simulation experiment and then performed two field tests, one on the campus of Nagoya University and the other at the Tsukuba Real World Robot Challenge. Our results showed that OpenPlanner is able to plan trajectories through complex vector maps and navigate through dynamic environments while handling a variety of discrete behaviors such as stopping at stop signs and traffic lights and avoiding obstacles. Without further optimization, OpenPlanner can function in real time at more than 10 Hz, even when number of obstacles increases to around 100. OpenPlanner can also generate very smooth trajectories, which allows for much smoother control.

The main contribution introduced in this chapter is to provide an integrated planner as an open source resource for the robotics community to use and enhance. The

source code is available as a collection of ROS packages within the Autoware project. It can be used as a stand-alone package or within the Autoware framework. It can use .kml format RNDF map files, which can be easily created and modified, and Autoware supported vector maps. The basic functionality of OpenPlanner is available as shared libraries, thus users can use it for development outside the ROS environment.

In this study we demonstrated that OpenPlanner can operate effectively and safely in dynamic environments by using a modified motorized scooter to successfully navigate through the Tsukuba Real World Robot Challenge and by navigating similar environments on the Nagoya University campus. Successful autonomous navigation requires avoidance of moving objects and pedestrians, the ability to follow moving objects along a path, navigating through narrow corridors, negotiating automatic doors, and stopping for traffic lights and stop signs. In our field tests, OpenPlanner demonstrated its ability to perform all of these tasks.

Chapter 4

Behavior planner based intention and trajectory estimation

4.1 Introduction

4.1.1 Problem Definition

Predicting with a high level of confidence what other vehicles are doing is essential for autonomous driving, and is one of the basic functions of a successful planner. Actions of other vehicles also include their probable intentions and future trajectories. The planner uses this information to generate suitable plans, i.e., ego-vehicle actions and trajectories.

In Section 2.4, we reviewed the literature to find a solution for the intention estimation problem. Three main approaches were discussed; deterministic modeling, machine learning and probabilistic modeling. Because it is essential to model uncertainties, we rejected the deterministic approach. In addition, we couldn't use machine learning due to a lack of data for all possible driving situations, especially the rarer ones. Finally, we determined that a probabilistic approach fits our problem domain, and chose to employ a particle filter. Several points should be kept in mind when choosing an intention estimator:

- It is important to find a reliable approximation to driving distribution (esti-

mation prior), i.e., able to solve $P(x_t|u_t, x_{t-1})$ where x_t is the state, u_t motion model, and x_{t-1} is the previously estimated state.

- Due to the complex nature of driving states, which contain a mixture of continuous and discrete variables, a conventional particle filter is not suitable.
- If a state's dimensions are too high, thousands of particles will be needed to represent the distribution, leading to much slower performance.

4.1.2 Contributions

The main contribution of this work is the development of a new method for estimating the intentions and trajectories of surrounding vehicles which can accurately handle most complex urban driving situations in real time. We accomplished this by using a behavior planner as the complex motion model, and integrating it with non-parametric probabilistic filter (a multi-cue particle filter) to handle uncertainty. Figure 4-1 highlights the main issues with the conventional approach and shows how we solved these problems.

Challenges using a conventional particle filter for estimation	Contributions of proposed approach
Difficult to model the complex motion parameters of surrounding vehicles.	Behavior planner is used as the motion model.
The state consists of both continues and discrete variables.	Multi-cue particle filter is used with variable confidence factors.
Since the state dimensions are high, thousands of particles are needed to capture the posterior distribution.	Separate particle filters are used for each intention and trajectory combination.
Multi-cue particle filter uses joint distribution to aggregate the weights of different sensing cues, leading to particle deprivation.	Weighted sum of different cues is used instead of the joint distribution.

Figure 4-1: Issues with conventional particle filter estimation algorithms and contributions of the proposed estimation method by addressing these problems.

4.1.3 Solution Approach

'Intention' and 'behavior' are generally used interchangeably to describe a driver's actions, which includes doing nothing (e.g., waiting for a traffic light to change), to

executing the complex, three-step maneuver used when passing a slower vehicle. In this work, we use 'behavior' to refer to the ego vehicle's actions or to the estimated actions of other vehicles before filtering. When we use 'intention', we mean the estimated set of actions after filtering [55][69].

Environment representation, such as representing custom-built road networks which provide some driving rules in a simulated environment, is very important for state modeling. In this work, we used the standard road network map format used in the full, autonomous driving software stack [19].

In this chapter, we describe our intention and trajectory probability estimation algorithm, which utilizes a behavior planner [25] working in passive mode, wrapped in a multi-cue particle filter [56] for uncertainty modeling.

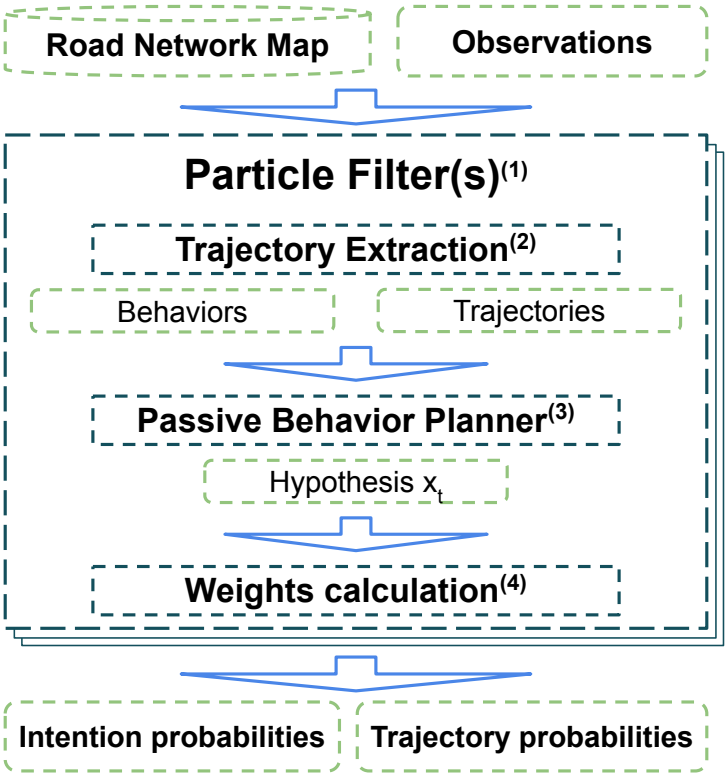


Figure 4-2: Proposed intention and trajectory estimation system. Multi-cue particle filters use a passive behavior planner as a motion model. Step (1) is explained in Section 4.3, Step (2) is explained in Section 4.4.1, Step (3) is explained in Section 4.2, and Step (4) is explained in section 4.4.3.

Figure 4-2 shows the system architecture of our proposed intention and trajectory

estimation system. The proposed algorithm consists of three main parts; a trajectory extractor, a passive behavior planner and multi-cue particle filters. Road network maps were explained previously in Section 3.2.1. Observations are the tracked objects, which are the output of a third-party object detection and tracking module. Finally, the algorithm outputs intention and trajectory probabilities which will be utilized later by the local planner. A detailed description of this system is provided in this chapter. First, the estimation algorithm extracts all of the possible driving trajectories and their embedded traffic rules from a road network map. Second, particles are sampled from the motion model function, which for this approach is the passive open planner. Third, weights are calculated for each sampled particle using the observations. Finally, the weights are normalized and the probabilities for each intention and trajectory are calculated.

4.2 Passive Behavior Planner

Recently developed behavior planners, such as OpenPlanner (discussed in Chapter 3), can generate deterministic behaviors for a vehicle to execute. The behavior planner represents other vehicles by estimating their internal parameters. The more accurately the planner can model the actions of the other vehicles, the better the estimation results that can be achieved.

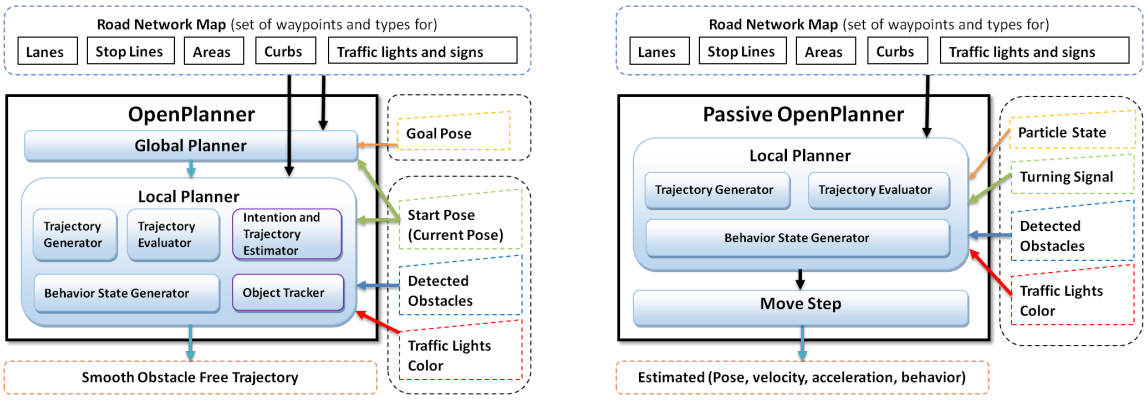
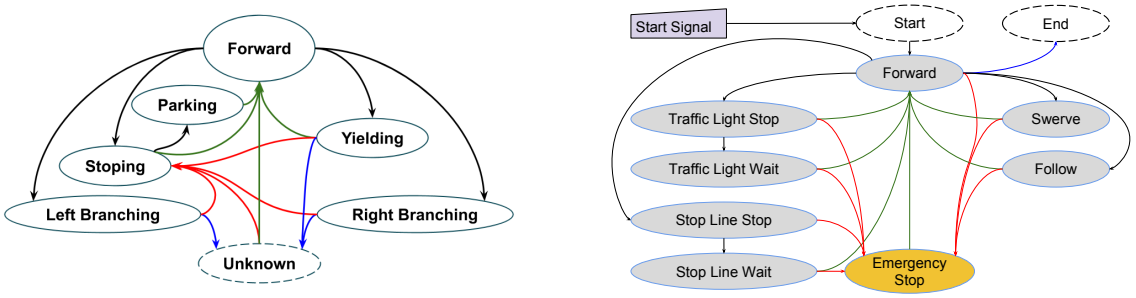


Figure 4-3: Comparison of our original behavior planner, OpenPlanner (a), to our modified behavior planner, Passive OpenPlanner (b).

The passive behavior planner shown in Figure 4-2 is a stripped-out version of the integrated planner described in Chapter 3. Figure 4-3 provides a comparison of the original OpenPlanner and Passive OpenPlanner. The integrated planner is so flexible that we were able to take out unnecessary modules such as the Global Planner, Intention Estimation and Object Tracking. It is called a passive planner because no feedback is available, thus the control signal doesn't have a direct impact on the observed state, and it has become similar to an open loop planner.

Another difference is that integrated planner output trajectory, then this trajectory is sent to a path following module to generate the desired control signal for the vehicle. But the passive behavior planner does that internally by simulating a **'move step'** which output directly the control signal velocity v, acceleration a, behavior beh, and the expected current position $pose$. Figure 4-4 (a) shows the intention state we are estimating. For each intention state in Figure 4-4 (a) there should exist a behavior that models the intention inside the passive behavior planner, Figure 4-4 (b). Multiple behaviors can model single intention and vice versa.



(a) (b)

Figure 4-4: Comparison of intentions to be estimated (a), to behaviors modeled by the passive behavior planner (b).

4.3 Uncertainty Modeling using Particle Filter

The basic idea behind particle filtering is to approximate the belief state $bel(x_t)$ at time t by a set of weighted samples χ_t as in Equation 4.1 [53]. x_t is the state vector, where z is the observation vector, χ_t^i means the i'th sample of state x_t. Equation

4.1 enable computing the posterior using importance sampling. Because it is hard to sample from the target distribution, we sample from a proposal or importance distribution $q(x)$, then weight the samples according to 4.2.

$$P(x_t|z_{1:t}) \approx \sum_{i=1}^{N_s} w_t^i \delta(x_t, \chi_t^i) \tag{4.1}$$

$$w^i \propto \frac{P(x_{1:t}^i|z_{1:t})}{q(x_{1:t}^i|z_{1:t})} \tag{4.2}$$

In particle filter the sample of a posterior distribution is called particles and denoted χ in definition 4.3 with M is the maximum number of particles. each particle $x_t^{[m]}$ is a concrete instantiation of the state at time t.

$$\chi_t := x_t^{[1]}, x_t^{[2]}, ..., x_t^{[m]} \tag{4.3}$$

Including the hypothesis x_t in the particle set χ_t require that it is proportional to the Bayes filter posterior believe $bel(x_t) = P(x_t|z_t, u_t)$ as In 4.4. Where z is the observation and u is the control signal.

$$x_t^{[m]} \propto P(x_t|Z_t, u_t) \tag{4.4}$$

The particle importance factor is denoted $w_t^{[m]}$. The particle weight $w_t^{[m]}$ is the probability of the measurement z_t under the particle $x_t^{[m]}$ and is given by Equation 4.5. Thus the set of weighted particles approximate the Bayes filter posterior $bel(x_t)$.

$$w_t^{[m]} = P(z_t|x_t^{[m]}) \tag{4.5}$$

4.3.1 General Particle Filter Algorithm

The general particle filter algorithm is shown in Algorithm 1. where χ_{t-1} is the particle set from the previous iteration. u_t is the most recent control signal. z_t is the most recent observation. m is the current process particle from particle set χ_{t-1} with size M. $x_t^{[m]}$ is the hypothetical state for current time step t and it is generated from the m'th particle in χ_{t-1}.

Algorithm 1 *Particle_Filter*(χ_{t-1}, u_t, z_t)

1: $\bar{\chi}_t = \chi_t = \phi$
2: **for** $m = 1$ *to* M **do**
3: *sample* $x_t^{[m]} \approx p(x_t|u_t, x_{t-1}^{[m]})$
4: $w_t^{[m]} = p(z_t|x_t^{[m]})$
5: $\bar{\chi}_t = \bar{\chi}_t + [x_t^{[m]}, w_t^{[m]}]$
6: **end for**
7: **for** $m = 1$ *to* M **do**
8: *draw i with probability* $\propto w_t^{[i]}$
9: *add* $x_t^{[i]}$ *to* X_t
10: **end for**
11: *return* χ_t

At the start in line 1 two empty sets of particles $\bar{\chi}_t$,χ_t are initialized to empty set ϕ. Then for each particle m in the particle set χ_{t-1}, we use its associated previous state $x_{t-1}^{[m]}$ to sample state $x_t^{[m]}$ and calculate the importance factor $w_t^{[m]}$. Then insert both the sampled state and the importance factor into the particle set $\bar{\chi}_t$.

Line 3 represent the sampling step of $x_t^{[m]}$ from the state transition distribution $p(x_t|u_t, x_{t-1}^{[m]})$. The importance factors w_t are used to incorporate the measurement z_t into the particle set. In Line 4 the particle weight $w_t^{[m]}$ is calculated using the probability of the measurement z_t under the sampled state $x_t^{[m]}$.

The actual trick for particle filter is the resampling step, Lines 7 to 10. For each particle m in the particle set $\bar{\chi}_t$ we draw a particle $x_t^{[i]}$ with probability proportional to its weight $w_t^{[i]}$ and add this particle to the resulting particle set χ_t. The main objective of this step is to remove particles with low importance, which move the particles distribution away from previous belief to become closer to the target belief.

Table 4.1: State space variable description

Variable	**Description**
$pose(x, y) \in R^2$	Position vector in meters
$\theta \in R$	Direction (heading) θ in radians
$v \in R$	Velocity in meters, only positive
$a \in R$	Acceleration in meter/second
$T.S \in \{-1, 0, 1, 2\}$	Turning signal $\in \{None, Left, Both, Right\}$
$Traj \in Z$	Trajectory index, from all possible driving trajectories
$Beh \in Z$	Behavior state index, from all behaviors supported in the behavior planner such as Forward, Stop, Yield, Park, Right and Left

Table 4.2: Observation variable description

Variable	**Description**
$pose(x, y) \in R^2$	Position vector in meters
$\theta \in R$	Direction (heading) θ in radians
$v \in R$	Velocity in meters, only positive
$a \in R$	Acceleration in meter/second
$T.S \in \{-1, 0, 1, 2\}$	Turning signal $\in \{None, Left, Both, Right\}$

4.3.2 Custom Particle Filter using Passive Behavior Planner

First it is important to define the problem's state space vector x and observation vector z. Table 4.1 describes the state vector x.

Observations could come from third party modules, such as object tracking and turn signal detection. Table 4.2 shows description of the utilized observations.

The main modifications introduced is the use of a deterministic behavior planner (Passive behavior planner) as the state transition distribution to generate the hypothetical state $x_t^{[m]}$ replacing line 3 in Algorithm 1 with Equation 4.6. The behavior planner is assumed to function as an expert driver who will follow the traffic rules as indicated by the road network map. This also enables us to use the simple motion model estimated by the behavior planner, thus replacing the behavior planner with a better one should result in a better model.

$$x_t^{[m]} \simeq PassiveBehaviorPlanner(z_t, x_{t-1}, map) \tag{4.6}$$

Multi-cue particle filter allow the use of several weak measurement cues to be accumulated into a strong estimator. It is flexible to add more sensing cues later for better estimation. The modification to the general particle filter algorithm 1 is in line 4, we replace the weight likelihood function with multi-cue one 4.7. In this technique we used measurement cues such as position *pose*, direction *dir*, velocity *vel*, acceleration *acc* and turning signal *sig*. The importance factor α used to keep the weight normalized where $\alpha_{pose} + \alpha_{dir} + \alpha_{vel} + \alpha_{acc} + \alpha_{sig} = 1$. More detailed about the weight calculation in section 4.4.3. The modifications introduced to the general particle filter algorithm is presented in Algorithm 2.

$$\begin{aligned} w_t^{[m]} = p(z_t|x_t^{[m]}) = \alpha_{pose}.P(z_{pose,t}|x_t^{[m]}) + \alpha_{dir}.p(z_{dir,t}|x_t^{[m]}) + \alpha_{vel}.p(z_{vel,t}|x_t^{[m]}) \\ + \alpha_{acc}.p(z_{acc,t}|x_t^{[m]}) + \alpha_{sig}.p(z_{sig,t}|x_t^{[m]}) \end{aligned} \tag{4.7}$$

Algorithm 2 $Modified\_Particle\_Filter(\chi_{t-1}, z_t, map)$

1: $\bar{\chi}_t = \chi_t = \phi$
2: **for** $m = 1$ *to* M **do**
3: *sample* $x_t^{[m]} \approx PassiveBehaviorPlanner(z_t, x_{t-1}, map)$
4: $w_t^{[m]} = \sum^{cue} p(z_t|x_t^{[m]})$
5: $\bar{\chi}_t = \bar{\chi}_t + [x_t^{[m]}, w_t^{[m]}]$
6: **end for**
7: **for** $m = 1$ *to* M **do**
8: *draw i with probability* $\propto w_t^{[i]}$
9: *add* $x_t^{[i]}$ *to* X_t
10: **end for**
11: *return* χ_t

A separate particle filter for each trajectory and intention needed to be estimated is used. This improves performance by reducing the dimensions of each particle filter. From Table 4.1 state dimension is D^8 which needs a very large number of particles.

We were able to reduce this number to D^6. In additional to improve the performance having separate particle filters enable use to control the particles population. In the resample steps we remove the particles with insignificant weight, that could lead to particle impoverishment. Minimum number of particles is essential to keep all intentions represented in the algorithm results.

4.4 Estimation Algorithm

The proposed algorithm consists of three steps; Trajectory extraction (Initialization), particle sampling and measurement update (weight calculation).

4.4.1 Trajectory Extraction (Initialization)

For the detected vehicle, all possible driving trajectories from the road network map is extracted. Then two additional trajectories to represent branching right and branching left are added. In Figure 4-5 the pink box shows the detected vehicle. Left side shows the center lines of the road network map, while the right side shows the four extracted trajectories; the two branching trajectories, one for going straight and another one for turning left. The long path represents the global plan for the simulated vehicle. When a new trajectory appears, a new set of particles are sampled from the initial prior distribution in 4-6.

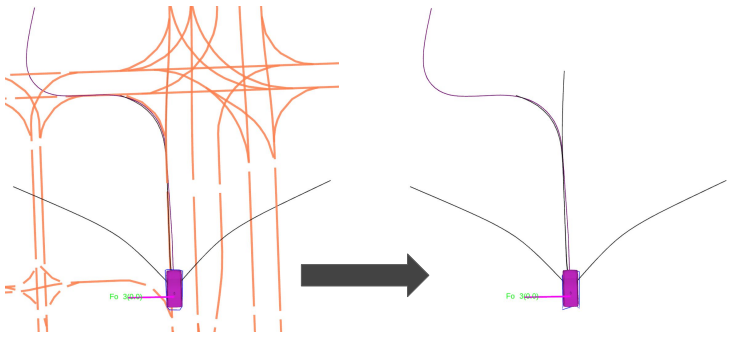


Figure 4-5: Extracting all possible vehicle trajectories from a road network map. After all of the existing paths are extracted, the additional branching paths (right) are added.

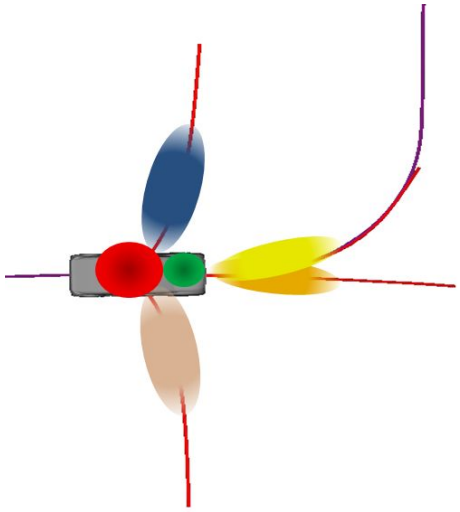


Figure 4-6: Prior sampling distribution, manually designed with fixed parameters for the particle filter's motion model initial condition.

4.4.2 Particle Sampling

Particle sampling follow line 3 in Algorithm 1. In this step we trying to create hypothesis particle state x_t^m using the state transition $p(x_t|u_t, x_{t-1}^{[m]})$. State transition probability follow the motion assumed for each particle. Usually simple motion model such as vehicle kinematics are used. It is important that this distribution presents the previous believe.

The passive behavior planner works as expert driving that models the internal parameter of driving process. So it can represent the distribution of the state transition better than simple motion model, including more complex driving scenarios. For that reason we used the passive behavior planner as the motion model, see modified Algorithm 2 line 3.

The behavior planner takes into consideration road network map and other observed vehicles. Simply for each particle the behavior planner calculate state x_t from x_{t-1} by using the map and current observations as in Equation 4.6.

For example if the detected vehicle is 10 meters away from a stop line at an intersection, and the previous state $x_{t-1}^{[m]}$ is similar to Table 4.3 column $t-1$. From the passive behavior planner point of view this vehicle should stop. So the state x_t for particle m will look like column $t(stop)$ in Table 4.3. Column $t(nostop)$ shows the

Table 4.3: Sampling step explained by example, state transition from $x_{t-1}^{[m]}$ to $x_t^{[m]}$

x_{t-1}	$x_t, stopsign$	$x_t, nostopsign$
$v = 5$	$v \approx 4$	$v \approx 5$
$a = 0$	$a \approx -1$	$a \approx +0$
$sig = None$	$sig = None$	$sig = None$
$Beh = Forward$	$Beh = Stop$	$Beh = Forward$

state transition if there is no stop sign in the intersection.

4.4.3 Weight Calculation

In this step the sampled particles are filtered by calculating how far the hypothesis distribution is from the measurement sensing cues. The weight for each particle is calculated against the sensing data, such as position, direction, velocity, acceleration and turn signal information using Equation 4.8. It is important to take sensing noise into consideration. We can use the same sensing noise value for each measurement, but that will not be accurate because the sensors vary in their degree of precision. But we can incorporate that noise into weight calculation 4.8 by replacing the importance factor α for each sensing cue with the sensing variance or noise model. Importance factors α must satisfy the condition that $\sum^{cue} \alpha = 1$.

$$w_t^{[m]} = \alpha_{pose}.w_{pose,t}^{[m]} + \alpha_{dir}.w_{dir,t}^{[m]} + \alpha_{vel}.w_{vel,t}^{[m]} + \alpha_{acc}.w_{acc,t}^{[m]} + \alpha_{sig}.w_{sig,t}^{[m]} \tag{4.8}$$

Position Cue Weight

Equations 4.10 and 4.9 represent the distribution $P(z_{pose,t}|x_t^{[m]})$. $w_{pose,t}^{[m]}$ is the distance difference between observed position z_{pose} and particle position $x_{pose,t}^{[m]}$.

$$w_{pose,t}^{[m]} = \begin{cases} \dfrac{1.0}{d}, if d > pose_{err} \\ \dfrac{1.0}{pose_{err}} \end{cases} \tag{4.9}$$

$$d = \sqrt{(x_z - x)^2 + (y_z - y)^2} \tag{4.10}$$

Direction Cue Weight

Equation 4.11 represents the distribution $P(z_{dir,t}|x_t^{[m]})$. Direction weight $w_{dir,t}^{[m]}$ is calculated using the difference between both observed z_θ and sampled $x_{dir,t}^{[m]}$. Discontinuity in angle differences could be problematic, so a special function is used in Equations 4.11 and 4.12 which contains information about vehicle heading semantics.

$$w_{dir,t}^{[m]} = \begin{cases} \dfrac{1.0}{|\theta_{diff}(z_\theta, \theta)|}, if|\theta_{diff}| > dir_{err} \\ \dfrac{1.0}{dir_{err}} \end{cases} \tag{4.11}$$

$$\theta_{diff}(\theta_1, \theta_2) = \begin{cases} 2\pi - (\theta_1 - \theta_2), if\theta_1 > \theta_2 \\ 2\pi - (\theta_2 - \theta_1), if\theta_2 > \theta_1 \end{cases} \tag{4.12}$$

Velocity Cue Weight

Equation 4.13 represents the distribution $P(z_{vel,t}|x_t^{[m]})$. Velocity weight $w_{vel,t}^{[m]}$ is calculated using the difference between both observed by object tracker z_{vel} and sampled velocity using passive behavior planner $x_{vel,t}^{[m]}$.

$$w_{vel,t}^{[m]} = \begin{cases} \dfrac{1.0}{|z_{vel} - v|}, if|z_{vel} - v| > vel_{err} \\ \dfrac{1.0}{vel_{err}} \end{cases} \tag{4.13}$$

Acceleration Cue Weight

Equation 4.14 represents the distribution $P(z_{acc,t}|x_t^{[m]})$. Acceleration weight $w_{acc,t}^{[m]}$ is calculated using the difference between both observed by object tracker z_{acc} and sampled acceleration using passive behavior planner $x_{acc,t}^{[m]}$.

$$w_{acc,t}^{[m]} = \begin{cases} \dfrac{1.0}{|z_{acc} - a|}, if|z_{acc} - a| > acc_{err} \\ \dfrac{1.0}{acc_{err}} \end{cases} \tag{4.14}$$

Turning Signal Cue Weight

Equation 4.15 and 4.16 represents the distribution $P(z_{sig,t}|x_t^{[m]})$. Turning signal weight $w_{sig,t}^{[m]}$ is calculated using the difference between both observed z_{sig} and sampled $x_{sig,t}^{[m]}$. If both values match we assign only 0.9 to the weight. We assume that there is 0.1 noise attached to the turning signal measurements.

$$w_{sig,t}^{[m]} = S_{diff}(z_{sig}, S) \tag{4.15}$$

$$S_{diff}(S_1, S_2) = \begin{cases} 0.9, if S_1 = S_2 \\ 0.1, if S_1 \neq S_2 \end{cases} \tag{4.16}$$

Implementation considerations

There are multiple approaches for weight; in our experiments we tried several methods. Finally we selected equation 4.9-4.16 as they are the most computationally efficient and provide explanatory results. Theoretically equation 4.8 is supposed to calculate the joint probability of all sensing information [56]. One problem with using joint probability and multiply all probabilities is lots of particles doesn't lie in the vicinity of the correct state. That leads to particle deprivation which requires the use of huge number of particles. Another problem is when one sensing cue importance factor becomes very low, other sensing weights contribute much less. To solve both problems, in equation 4.7 and 4.8 we used aggregation of the weights instead of the joint probability with the assumption that all sensing cues are independent.

After weight calculation we normalize the particles so that $\sum_m w_t^{[m]} = 1$. Then particles with small weight are removed and are replaced with new samples drawn from particles with the highest confidence level. This sampling step relies on the motion model that estimates the path the observed vehicle is expected to follow,

which in this study is the passive behavior planner. It estimates the motion and intentions of other vehicles according to the observed circumstances Equation 4.6.

4.4.4 Importance factor setup

Initially the importance factor α for each measurement cue distribution is the same $\alpha = 0.2$, satisfying the criteria $\sum^{cue} \alpha = 1$. But this importance factors could be utilized more efficiently. Sensors and sensing modules usually calculate variance or noise probability to express the reliability of its data. These variances could be mapped to the importance factor so when one sensing measurement is not reliable enough its importance factor should decrease, thus contribute less to the total weight. In our experiments we used the importance factor to disable some measurement cues manually. For example in section 4.5.2 to compare between results of having turning signal measurement or not, we set $\alpha_{sig} = 0$ and the other importance factors such that $\alpha_{pose} = \alpha_{dir} = \alpha_{vel} = \alpha_{acc} = 0.25$.

4.5 Evaluation Results

In this section we present a detailed analysis of the results of our estimation process. We evaluated our proposed intention and trajectory estimation system using multiple simulated driving situations. The objective of these evaluations is to demonstrate that the proposed method can accurately estimates trajectories and intention probabilities at various driving scenarios, such as three-way intersection, four-way intersection, intersection with and without stop signs, and bus stops. These driving scenarios are depicted from the National Highway Traffic Safety Administration (NHTSA) report [70].

The simulated vehicles operate using an active version of the behavior planner. State and generated behaviors for the simulated vehicles are considered as the ground truth. There is no information sharing between the planner for the simulated vehicles and the ego vehicle's planner. An additional perception simulation module received the ground truth for vehicle pose and size, which was then published in a similar

message format as the object detection algorithm. The perception simulator added random noise to the position and point cloud data before publishing. Starting conditions are fixed for each experimental trial in order to facilitate a comparison between the results. Finally, the ground truth from the simulated vehicle and the estimation results from our proposed method are compared to measure performance. All simulations used actual intersections from the road network map of the area around Nagoya University.

Two values are used to discriminate intentions and trajectories. Average weight, which is the average of all of the particles' normalized weights, represents signal behavior at each time step. The average is calculated after the re-sampling step. The second value is probability, which is the ratio between the remaining particles (after elimination of the weak ones) to the maximum number particles. This probability is calculated before the re-sampling step. Both measurements can be used in different ways, but usually average weight changes more smoothly over time.

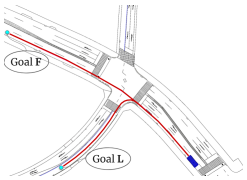
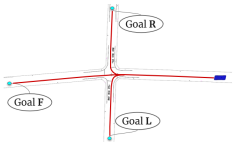
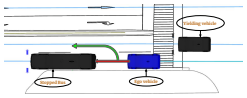
In the particle filter re-sampling step, weak particles are replaced with new particles. These new particles are randomly drawn from the normal distribution around the previous best state. Sometimes when new particles are generated, the weight calculation spikes. After few iterations it converges to the general trend. Increasing the number of particles minimize these spikes but hurts the performance. In the experiments we conducted these changes (spikes) never big enough to affect the estimation process. Intentions is abbreviated in the graphs by the first character of each intention, Moving Forward **(F)**, Stopping **(S)**, Branching Left **(L)**, Branching Right **(R)**, Yielding **(Y)**, Accelerating **(A)**, and Parking **(P)**.

Table 4.4 shows a summary of all experiments discussed in this section.

4.5.1 Three-way Intersection

In this experiment, Figure 4-7, there are two possible trajectories the simulated vehicle can follow. Forward trajectory until the intersection and then continuing forward towards Goal **F**, or a forward trajectory until the intersection and then turning left towards Goal **L**. Assumptions for this experiments are that the traffic light is green

Table 4.4: Experiments.

Experiment	Environment	Objectives
Three way Intersection		- Shows weight calculation and estimated intentions and trajectories associated with different Goals
Four Way Intersection **(Goal F)** No Stopping, Turn Signal On		- Estimating the intention and trajectory of going forward assuming the detection of turning signal.
Four Way Intersection **(Goal F)** No Stopping, Turn Signal Off		- Estimating the intention and trajectory of going forward without using the turning signal measurement.
Four Way Intersection **(Goal F)** Stopping, Turn Signal On		- Estimating the intention and trajectory of going forward with stopping and using turning signal measurement.
Four Way Intersection **(Goal F)** Stopping, Turn Signal Off		- Estimating the intention and trajectory of going forward with stopping and excluding turning signal measurement.
Four Way Intersection **(Goal L)** Turn Signal On		- Estimating the intention and trajectory of going left including turning signal measurement.
Four Way Intersection **(Goal L)** Turn Signal Off		- Estimating the intention and trajectory of going left excluding turning signal measurement.
Bus Stop **(Parking)**		- Estimating the parking intention.
Bus Stop **(Overtake)**		- Estimating the yielding intention to generate overtake behavior.

and that turn signal sensing is not available.

Figure 4-8 shows the speed profile and the state transition for the planner generated behaviors. The target intentions are mapped to the velocity profile graph. Figure 8 shows the average weights of the particles associated with each intention (Forward, Stop and Yield). The most probable intention to be selected by the planner at each time step for Trajectory **L** is shown in Figure 4-10. For Trajectory **F**, Intention weights associated with it is shown in Figure 4-10.

Trajectory probability estimation can be achieved either by averaging the weights

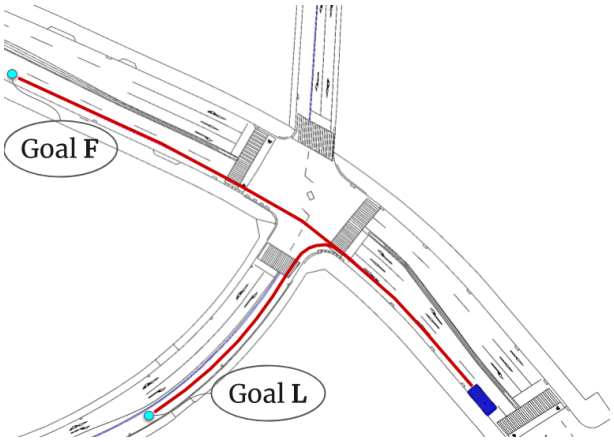


Figure 4-7: Simulated vehicle paths. Red dotted lines represent the global paths for the simulated vehicle. Simulations run separately for Goal **F** and Goal **L**.

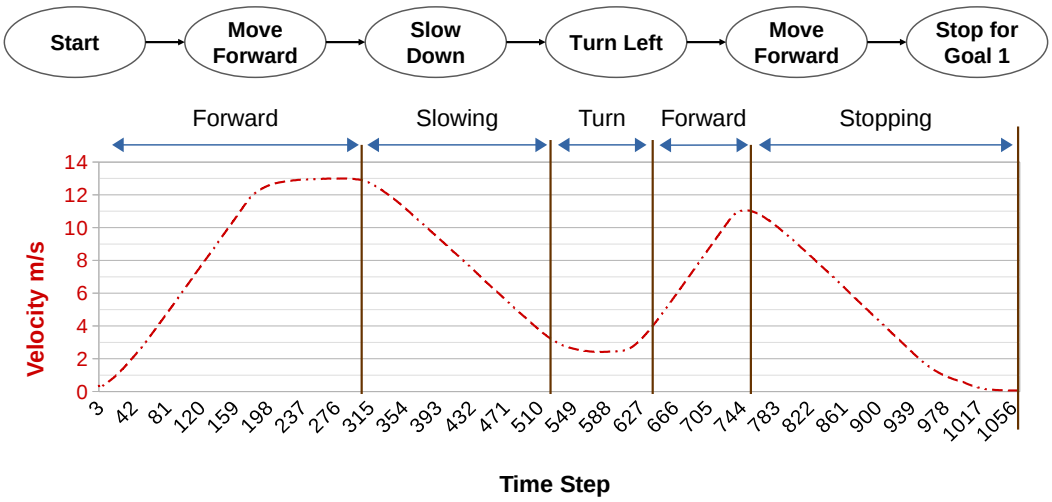


Figure 4-8: State transition and velocity profile of the simulated vehicle when moving toward Goal **L**

of all of the particles associated with the trajectory, or by only using the particles associated with the Forward intention. Intuitively, Forward intention weights should be sufficient to discriminate between trajectories, and the data from our experiments support that intuition.

4.5.2 Four-way Intersection

In this experiment, Figure 4-12, the simulated vehicle starts from the common starting position and moves towards one of three goals; Forward **(F)**, Left **(L)** or Right **(R)**.

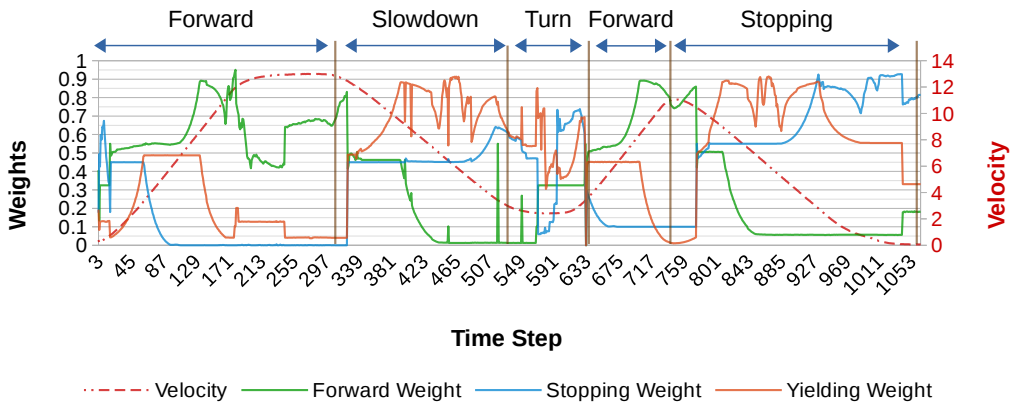


Figure 4-9: Data associated with Trajectory **L**, comparing the average weights to the ground truth velocity profile and behaviors.

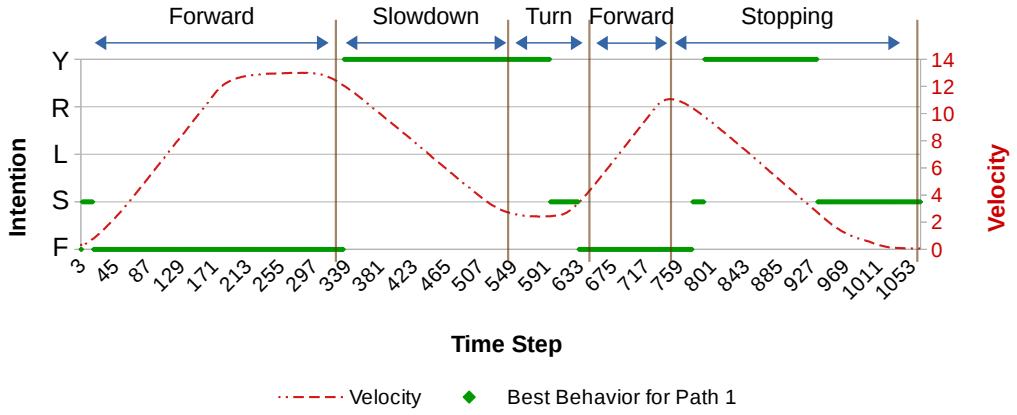


Figure 4-10: The most probable estimated intention at each time step for Trajectory **L**

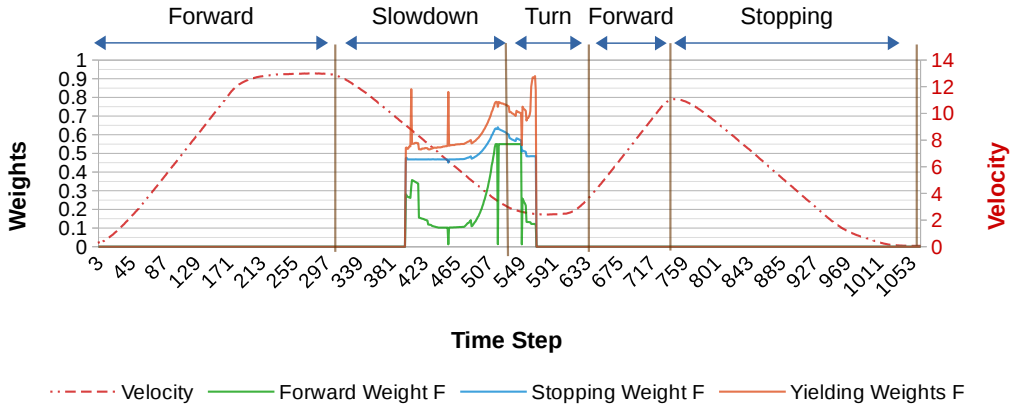


Figure 4-11: Data associated with Trajectory **F**, comparing the average weights to the ground truth velocity profile and behaviors.

The experiment repeats from starting position to each goal, and during each trial we enable or disable the ability to use turn signals as a sensing cue. In addition, we enable or disable stopping at the stop sign. The analysis of our results shows the effect of adding more sensing information on the accuracy of the resulting probabilities.

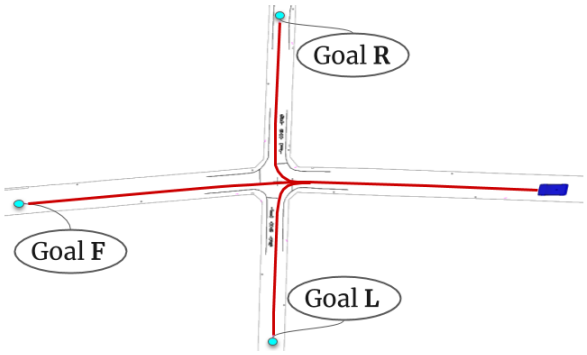


Figure 4-12: Testing scenario with a four-way intersection. The simulated vehicle starts at the position of the blue car model then moves toward one of three goals (**L**, **F**, **R**).

Goal F: No Stopping, turn signal sensing on

From the perspective of a human driver, it is highly probable that the vehicle will continue to move forward without stopping or slowing down if no right or left turn signal is observed and the vehicle is moving faster than turning speed. Thus, the estimated intention should be Forward and the estimated trajectory should be the 'go forward' trajectory, Trajectory **F**. Figure 4-13 shows the resulting average weights used to estimate the intention of the observed vehicle in Figure 4-14. At times, the weights are similar, so confidence when selecting an intention is low, but at other times it is clear that there is only one likely intention, although there is still a small possibility of other intentions. The critical intersection zone is indicated in the graph, approximately between the 150th and 400th time steps. In Figure 4-13 it is obvious that the predicted intentions match the ground truth behaviors with high confidence most of the time, especially in the intersection zone. In this experiment, Trajectory **F** was predicted with high probability, as shown in Figure 4-15, where the probability of Trajectory **F** is about 0.8.

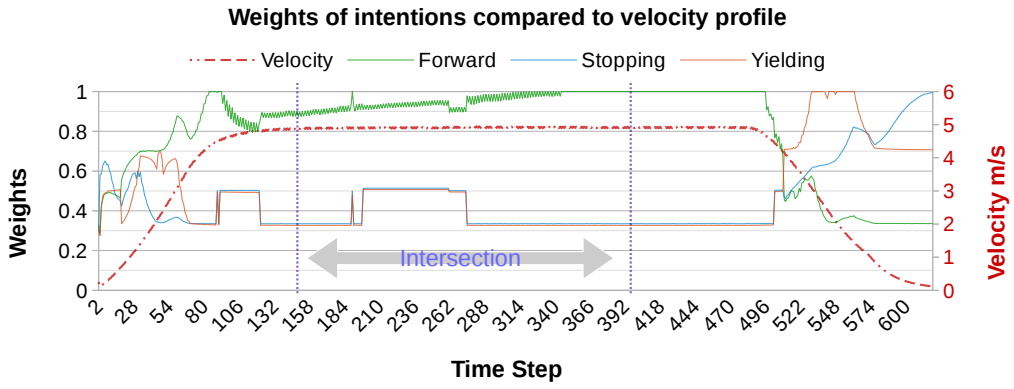


Figure 4-13: Average weight data associated with Trajectory **F**, including indicator sensing and not stopping for the stop sign.

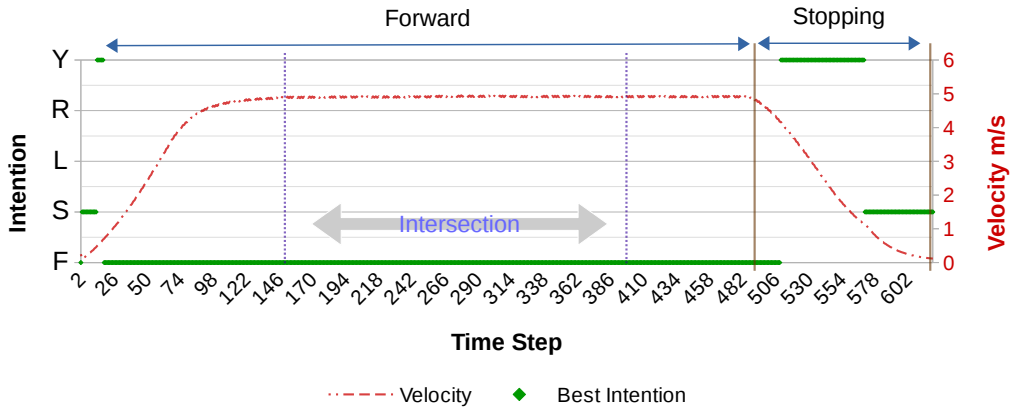


Figure 4-14: Most probable estimated intention at each time step, including turn signal data and not stopping at the stop sign.

Goal F: No Stopping, turn signal sensing off

Results in Figures 4-16 and 4-17 are the same as in the previous experiment, indicating that in this scenario turn signal information does not have much effect on the final estimation results. However, when we compare Figures 4-13 and 4-16, we see that the intention weights are slightly higher when turn signal information is included. The same relationship could be observed for trajectory estimation when comparing Figure 4-15 (with turn signal information), and Figure 4-18 (without turn signal information).

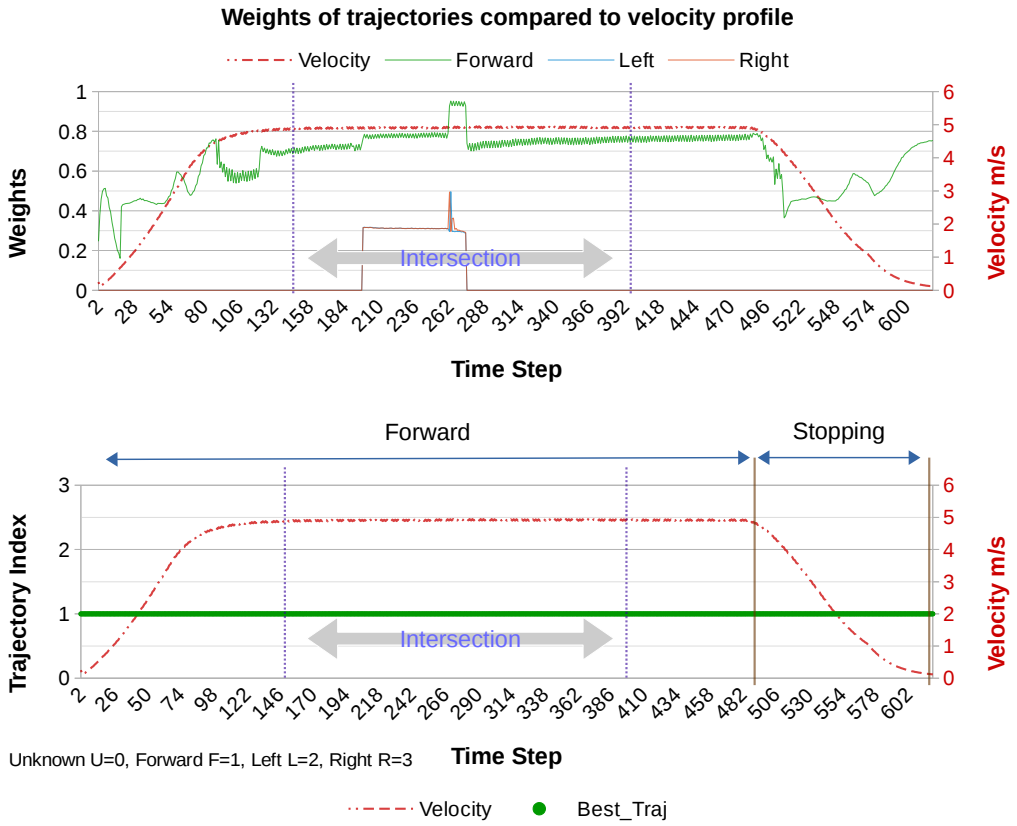


Figure 4-15: Trajectory estimation for moving towards Goal **F**, including turn signal data and not stopping for the stop sign.

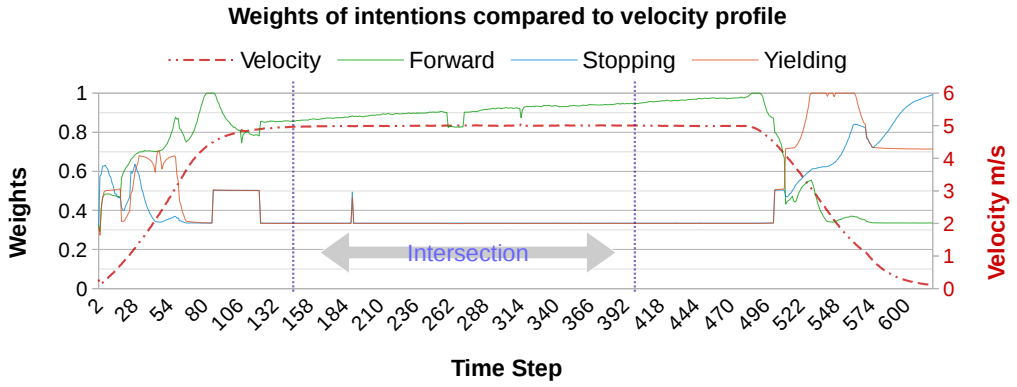


Figure 4-16: Average data weights associated with Trajectory **F**, when deactivating turn signal sensing and not stopping for the stop sign.

Goal F: Stopping, turn signal sensing on/off

Figure 4-19 shows the weights and selected intentions when turn signal information is not included. Figure 4-20 shows the weights and selected intentions when turn signal

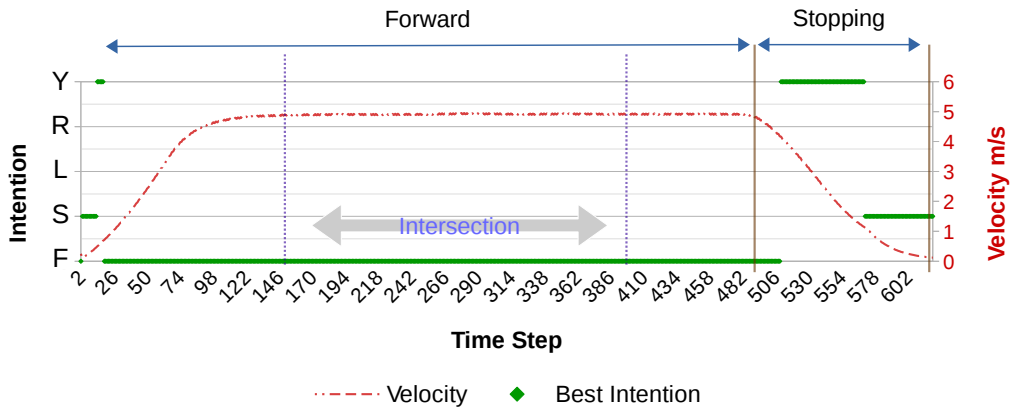


Figure 4-17: Most probable estimated intention at each time step, when not using turn signal information and without stopping.

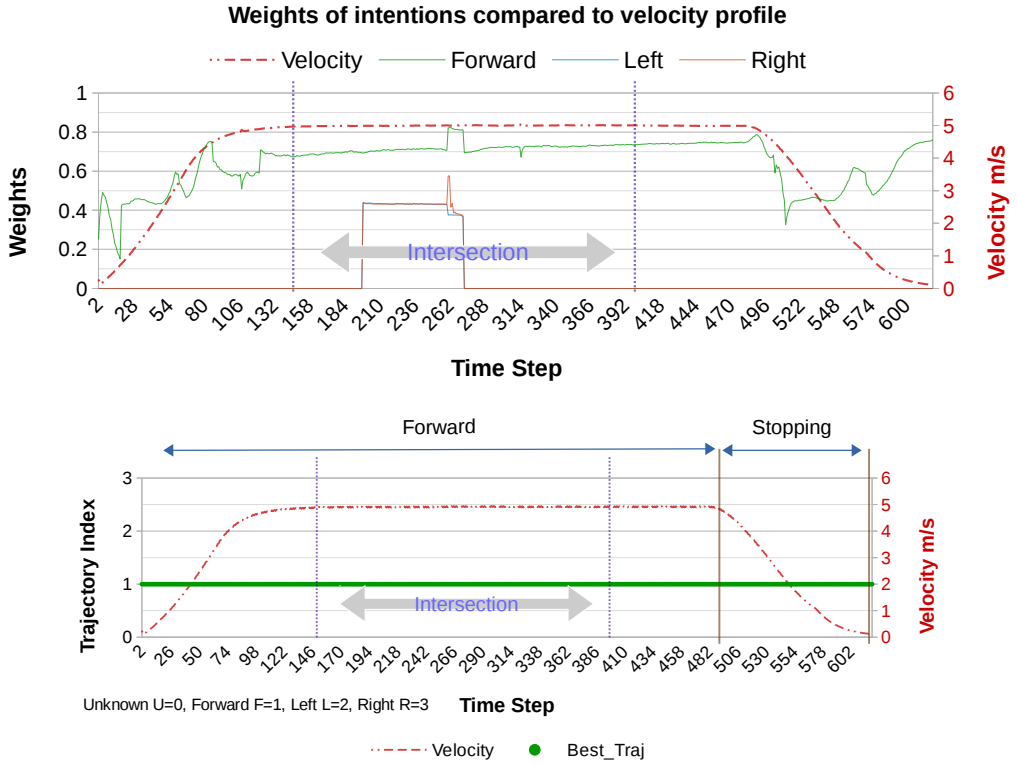


Figure 4-18: Trajectory estimation for moving towards Goal **F**, when not using turn signal data and not stopping for the stop sign.

sensing is included. We can see from the results how using the turn signal data affects the results.

When the vehicle stops at the intersection, we lose forward velocity as a clue for helping choose the most likely trajectory, so there is not enough evidence for the

particle filter to converge towards one of the three trajectories, especially when turn signal information is not taken into consideration. In Figure 4-21, we can see that the algorithm was confused and couldn't estimate the correct trajectory when the vehicle stopped for the stop sign, but that it estimated correctly once the vehicle started to move again. This problem is resolved by using the turn signal data, as shown in Figure 4-22, the prediction algorithm gives a higher confidence to the correct trajectory even though the vehicle is stopped. Although the algorithm selected the Forward trajectory as best choice, the Left and Right trajectories still have probabilities as high as the intended one. This could happen when the driver forgets to use the turn signal or changes his mind at the last moment.

Figure 4-19: Average weights and intention index data associated with Trajectory **F**, when **not** including turn signal data and stopping for the stop sign.

Figure 4-20: Average weights and intention index data associated with Trajectory **F**, when including turn signal data and stopping for the stop sign.

Goal L

This experiment is similar to section 4.3.1 experiment, Figure 4-23 compares the effect of using turn signal data on how early we can estimate the vehicle's trajectory. In the worst-case scenario (no turn signal data) the algorithm can still estimate the vehicle's trajectory accurately once the vehicle starts turning at the intersection (within 1 to 2 seconds).

Goal R

When conducting the Goal **R** experiment, we obtained similar result as in our Goal **L** experiment. We are able to obtain consistent estimation results, but we still could not predict the driver's intention when the vehicle stopped at the intersection without turn signal information. In fact, even human drivers cannot correctly predict which

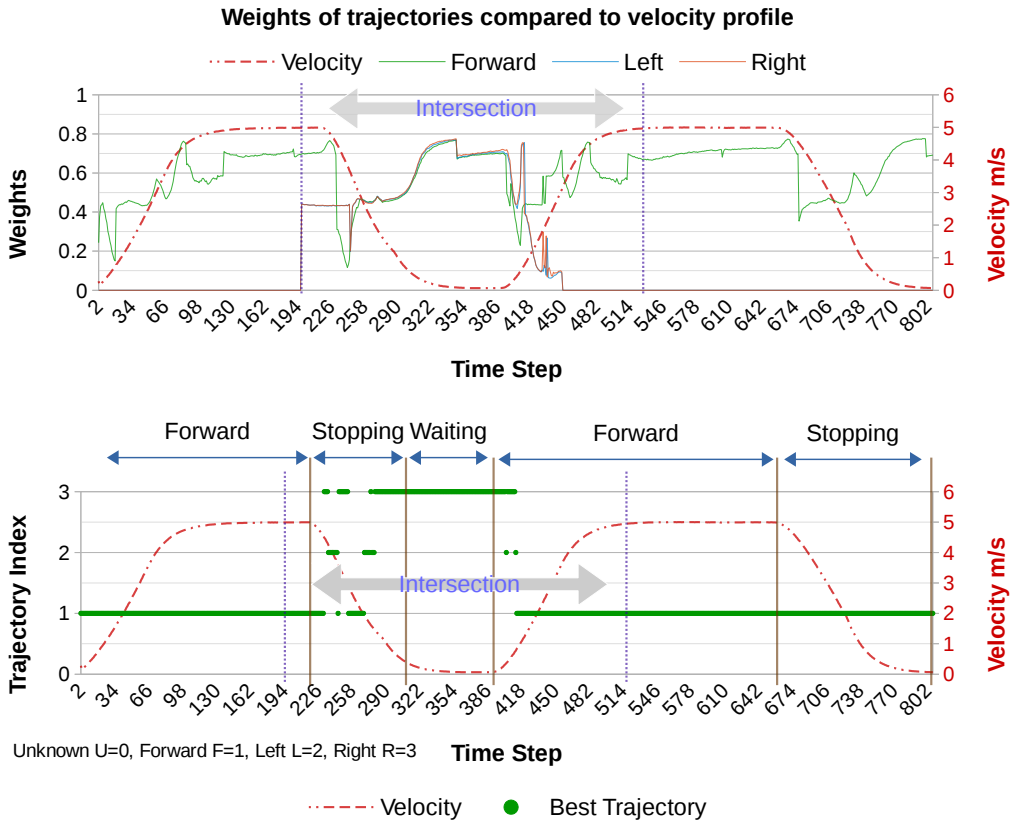


Figure 4-21: Trajectory estimation for moving towards Goal **F**, without turn signal data and stopping for the stop sign.

direction a vehicle will turn from a stop without turn signal information, until the moment that the vehicle starts to turn.

4.5.3 Bus Stop and Parking Scenario

Other frequent driving scenarios include attempting to avoid parked vehicles and buses waiting at bus stops. A simple rule that humans use is to observe and understand a driver's hazard lights vs. turn signals, i.e., whether the lights on both sides of the vehicle are flashing, or only on the right or left side. Another strategy is to determine if the vehicle is stopped by the side of the road without any stop line or traffic light nearby.

In Figure 4-24 the ego vehicle is coming from behind in the same lane as the bus. Another vehicle is moving forward in the adjacent lane, blocking the ego vehicle from

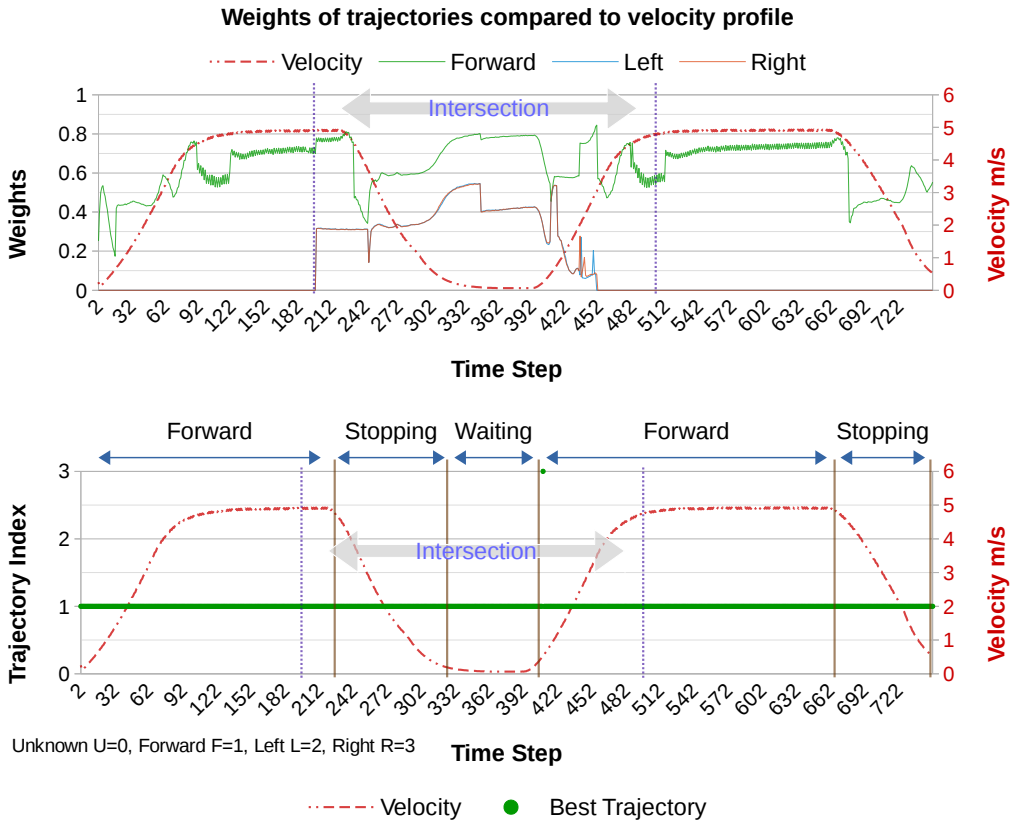


Figure 4-22: Trajectory estimation for moving towards Goal **F**, with turn on signal data and stopping for the stop sign.

passing the bus. The algorithm needs to estimate with high confidence that this bus is stopping and that the stop is not caused by other traffic circumstances, such as a traffic light, stop sign or normal traffic jam. To help handle such a scenario, we added a new intention (Parking). Figure 4-25 shows the estimated intentions of the bus against its tracked velocity profile. In this experiment hazard lights are used to differentiate general stopping from waiting by the roadside.

4.5.4 Overtaking Scenario

In case of passing the parked bus, ego vehicle's planner needs to identify if the vehicle in the adjacent lane is yielding or not. If a yielding intention is estimated with enough confidence, the ego vehicle can safely change lanes. Figure 4-26 shows the estimated intentions for the vehicle driving alongside the ego vehicle. The results indicate that

Figure 4-23: Comparison between weights associated with trajectories when moving towards Goal **L**, with (top) and without (bottom) turn signal data.

the vehicle has yielded to let the ego vehicle pass the bus. In this case, the ego vehicle planner will decide a safe lane change trajectory.

4.6 Conclusion and Summary

In this chapter we proposed an intention and trajectory estimation method for predicting the behavior of surrounding vehicles. The final goal of the proposed method

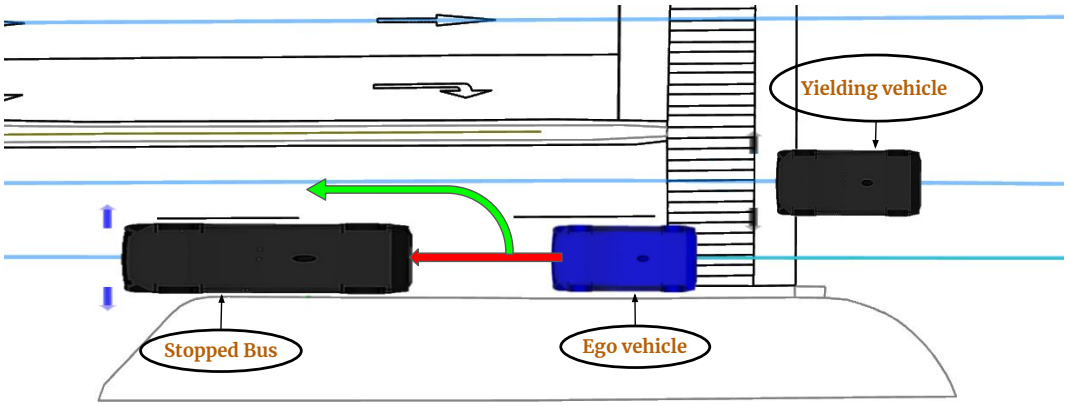


Figure 4-24: A bus is stopping at a bus stop. The ego vehicle is traveling in the same lane as the bus, while another vehicle is traveling in the adjacent lane.

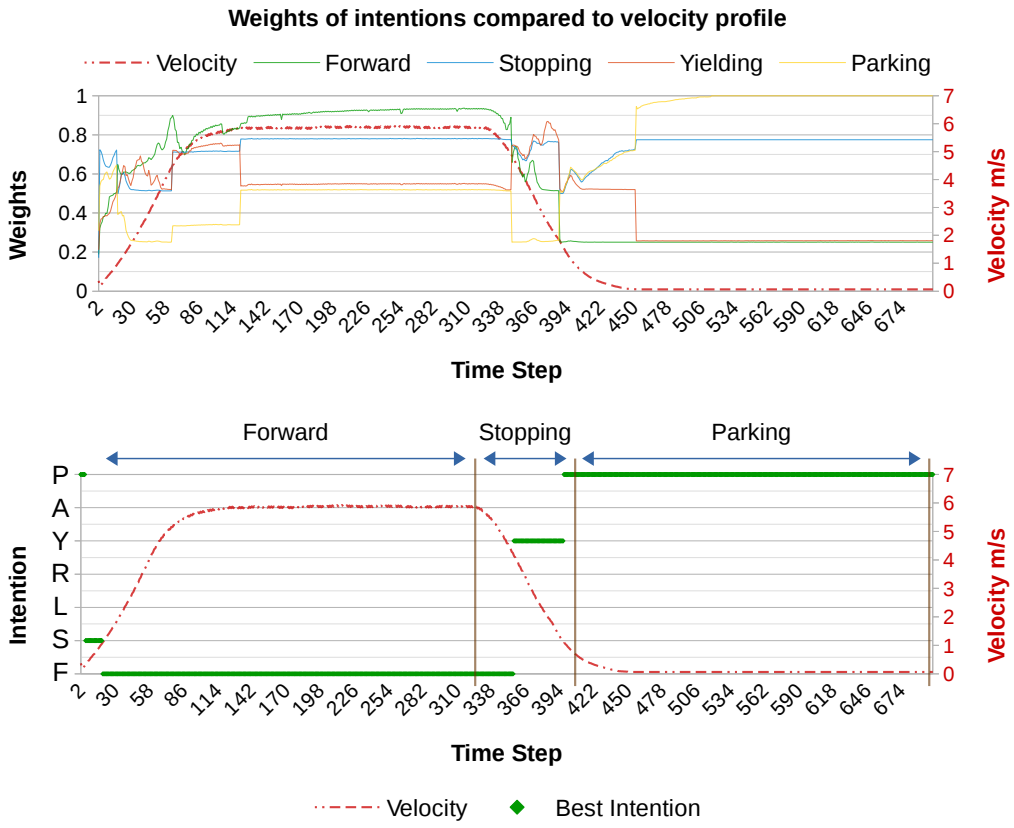


Figure 4-25: Data associated with a bus stopping at a bus stop. Top: weights associated with each intention. Bottom: most probable intention at each time step.

is to associate a probability with each intention and trajectory. This is a very important step before decision making in an autonomous driving system's planning process.

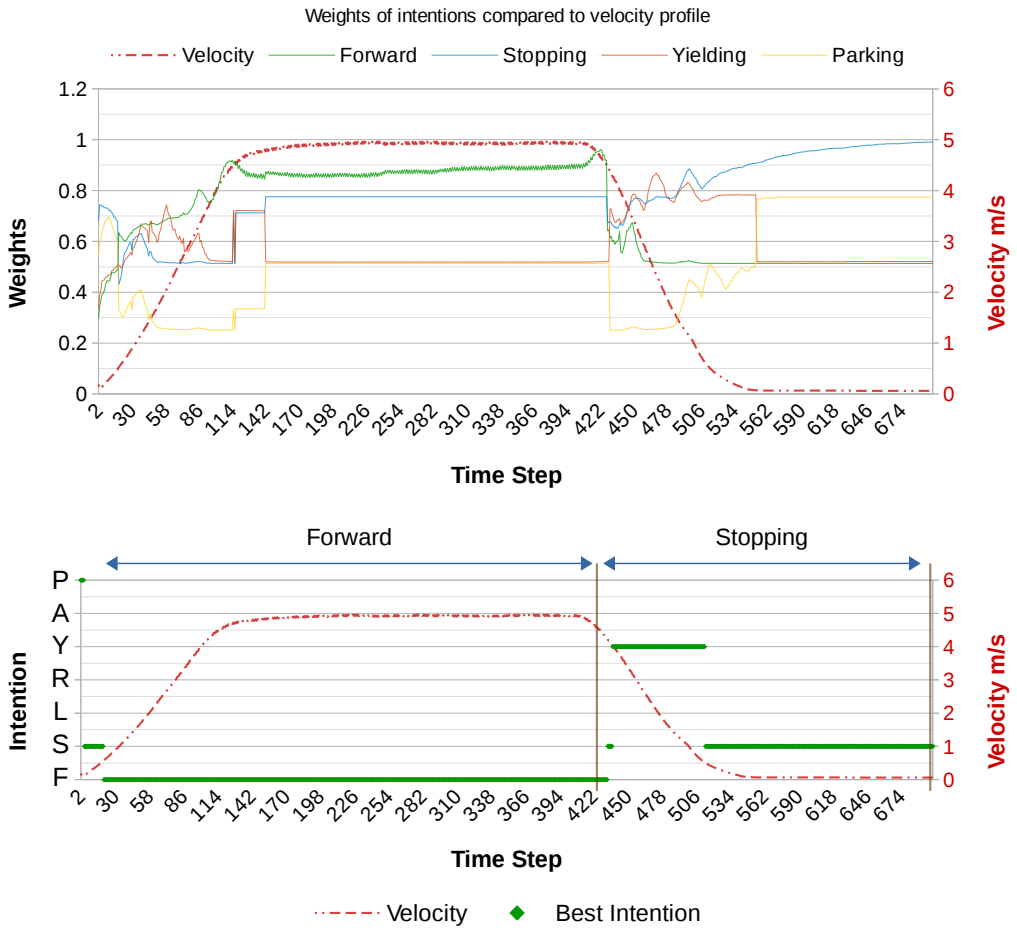


Figure 4-26: Data associated with the vehicle in the adjacent lane. Top: weights associated with each intention. Bottom: most probable intention at each time step.

Our method works by first associating a particle filter with each intention and trajectory. Then, a deterministic behavior planner is used to provide the expected motion model for each particle. After that, the particle weights are calculated by finding the difference between the expected state and the actual sensing information, using multiple sensing inputs such as position, velocity, acceleration and turn signal status. Our results show that our proposed method has the ability to accurately discriminate between various possible intentions and trajectories in a variety of complex driving situations. Contributions of this work are as follows:

- By using a behavior planner, we were able to model the intentions and trajectories of surrounding vehicles. Our results show that any new behavior the

planner supports, including 'socially aware', 'complex' or 'rare', can be easily integrated into the estimator, and that our model works reliably prior to the filtering step.

- By using a particle filter, we were able to address the problem of sensing uncertainty.
- Particle filter performance was improved by reducing particle state dimensionality. This was achieved by assigning a separate filter to each intention or trajectory.
- By using multiple sensing cues, our method allows earlier estimation and increases estimation confidence. The proposed method also has the ability to handle sensing problems such as variations in sensor accuracy, adding a new sensor or losing an existing sensor.

The next step is to integrate the estimated probabilities with the ego vehicle's behavior planner, in order to plan smoother velocity profiles and generate safer decisions. From the target application point of view, this work is targeting level 4 and level 5 Autonomy. More computational platform based optimization is needed to support level 3 active safety application. The algorithm design is flexible enough to improve the performance using multi processor or GPU based in vehicle computational platforms.

Chapter 5

Real World Data Circulation and Social Impact

In this chapter, I will discuss the work described in this dissertation and the concept of Real World Data Circulation (RWDC). In addition, I will introduce two examples which illustrate utilization of RWDC and explain their social value. The first example is a project to automate the creation road network maps, and the second is based on my experience working as an intern at an autonomous driving initiative project. The idea of RWDC is based on openly creating organizing and sharing data as raw data or well data sets, these data capture the problem enabling teams to explore more possibility offline, analyzing existing solutions and developing new ones based on the openly shared data. After development another set of data is created and shared which help bench-marking the problem's solution. Data circulation creates a modern and unique way for international collaboration to tackle the most challenging technological problems.

5.1 Autonomous Driving Planning

Previously in this dissertation, I have described the development of a complete, integrated, open-source planner for autonomous driving, the implementation of which is called OpenPlanner [25], a component of the Autoware [18] platform.

5.1.1 Contribution to Society

Our development of OpenPlanner, and its introduction as an open-source application, has so far achieved the following:

- Hundreds of users, as well as feedback from the autonomous mobility community.
- International collaboration with several different teams, working in different countries and focused on different goals.
- Experiment and data sharing between development teams to improve the platform and create safer autonomous driving systems.
- Use of our planner in multiple projects which directly improve daily life, such as the Roboat project [23]

5.1.2 Data Circulation

In Figure 5-1, a data circulation graph shows how our planner leverages RWDC to evolve and become more reliable. A variety of data is shared, such as sensing data, position data, driver-related data, environment-related data, HD maps and road network maps. Some logs become standard ROS bag files, while other logs are customized and require conversion tools. Cleaning, analyzing and organizing the shared data is also important in order to construct usable data sets for learning and bench marking. For example LIDAR data, sensor log is recorded while driving, these logs help creating HD maps, which are essential for localization, after that these HD maps are shared by another team. Another team uses the shared HD maps to collect driving data including images from camera, and then these driving logs are shared. Another team uses the maps and LIDAR data and image data to develop and test new algorithm, these algorithms is used to drive a vehicle in reality or in simulation and collect and share driving data, and so on.

Another example of data circulation is the open-source code itself. The code of the planning algorithm is itself data, in addition to being a generator and analyzer

of data. Figure 5-2 illustrates the idea of open-source application development as a form of data circulation.

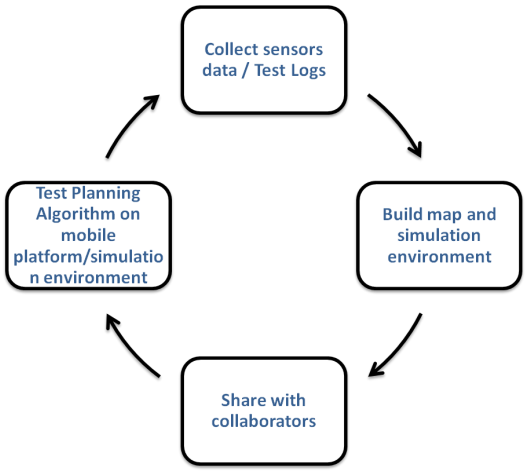


Figure 5-1: Relationship between OpenPlanner and RWDC.

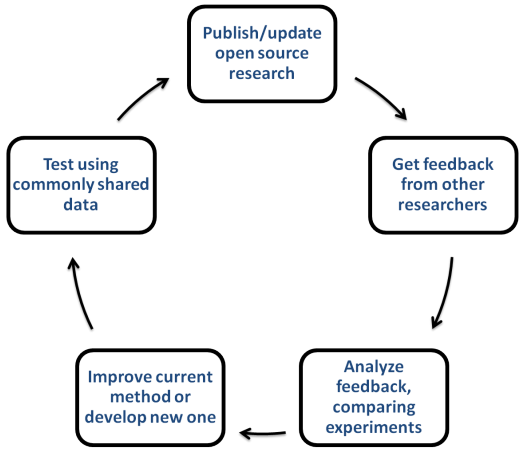


Figure 5-2: OpenPlanner and other open-source code projects as examples of RWDC.

5.2 Automated road network mapping (ASSURE Maps)

The Creative Research Project **(CRP)** is one of the main activities for the Leading Program at Nagoya University [71]. Every year, the Leading Program offers funding through the CRP to one of several proposed research projects. In 2017, our team proposed ASSURE Maps, which uses various types of driving data to automatically

create road network maps. In this section I will describe the ASSURE Maps project and its results.

ASSURE Maps was a contribution to the search for solutions to the autonomous driving planning problem. Most autonomous driving planners, including **OpenPlanner**, utilize road structure maps. High quality road network maps improve the accuracy and performance of planning dramatically, resulting in safer and smarter autonomous vehicles.

5.2.1 Introduction

Accurate road network maps are an essential part of reliable autonomous driving systems. Their structure is similar to the navigation maps widely used in modern cars, and they share many of the same features. Currently available road network maps are GPS based, thus their accuracy is within a few meters. Another limitation of current maps is their low level of detail. For example, GPS road network maps do not include traffic signs or lane marking locations, or the type of traffic lights.

Starting in 2007, the Defense Advanced Research Projects Agency's (DARPA) Urban Challenge introduced a simple road network definition file (RNDF) for teams to use to navigate through the competition course [3] [4]. Since then, improved RNDFs, and more recently HD road network maps or vector maps, are being used for public road testing of autonomous vehicles. These maps are still been created and annotated manually, using simple graphing tools, which is a slow and inefficient process that also suffers from frequent human errors.

Since there are millions of kilometers of roads which need to be accurately mapped, the current process is far too slow and expensive. Furthermore, there is a famous saying in the autonomous driving community: "Whoever owns the maps, owns the autonomous driving business". Since the majority of the roads globally do not have an accurate Road Network Maps yet, and since the current process for generating vector maps is slow, costly and prone to human error, our ASSURE Maps project focused on developing a method of automatically building HD road network maps for autonomous vehicles. The goals of the ASSURE Maps project (and the source of the

project's name) were as follows:

- **A**ccurate road network maps
- **S**ecure cloud service
- **S**mart mapping tools
- **U**pdated maps
- **R**ich details
- **E**valuated results

We can see from Figure 5-3 that the autonomous driving market will grow rapidly over the next 20 years. Therefore ASSURE Maps has much potential, since it fills a serious gap in the autonomous driving technology sector.

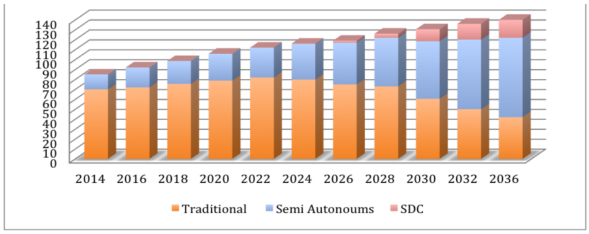
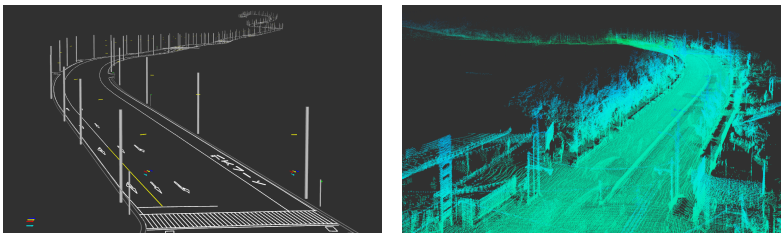


Figure 5-3: Business potential of autonomous driving (in millions of dollars).

Our project was focused on road network map generation, but there is another type of map currently being used for autonomous driving applications, which are known as point cloud maps. These maps are primarily used for detecting objects and features in the immediate driving environment. Figure 5-4 shows examples of both types of maps. Point cloud maps are generated using high quality Light Detection and Ranging (LiDAR) sensors mounted on vehicles. Road network maps, on the other hand, are essential for Level 4 and Level 5 autonomous driving planning, and crucial for safety since they allow autonomous systems to comply with road traffic rules. The majority of roads globally do not have accurate road network maps yet, as the current process of generating vector maps manually is lengthy, costly and inefficient, slowing the process of wider adoption of autonomous vehicles.



(a) (b)

Figure 5-4: Road Network map (left), Point cloud map (right).

5.2.2 Project Design

The ASSURE Maps system consists of two main modules. The first module uses automatic map generation **(AMG)** APIs, and the second uses smart mapping tools **(SMT)**, as shown in Figure 5-5. ASSURE's AMG APIs function as the internal engine that loads data logs (LiDAR, camera images, GPS data and odometry) and extracts map semantic information. The smart mapping tools module functions as a review tool which help users control the output of the AMG APIs and creates data sets for the system's machine learning-based components.

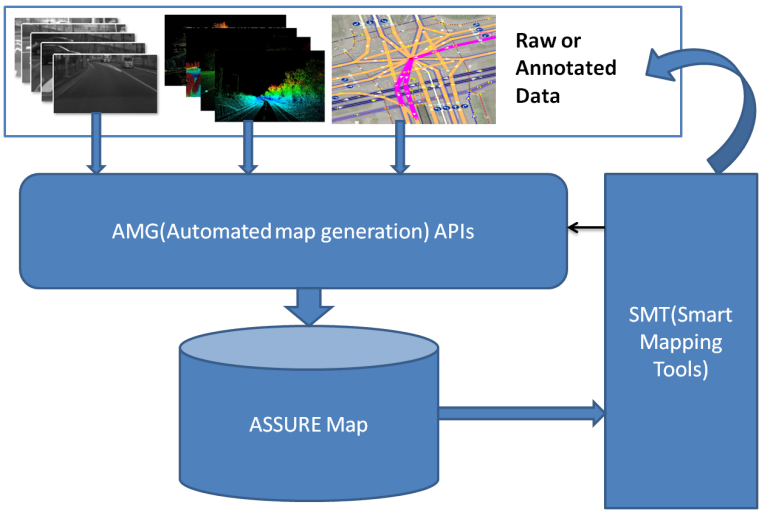


Figure 5-5: ASSURE Maps Architecture.

As shown in Figure 5-6, the system reads data from collected data sets, which can be saved as either a ROS bag file or as a collection of .csv files. The system needs at least a video camera and LiDAR sensors to build accurate maps. ASSURE smart tools load the data from the log files, start the AMG APIs to generate the maps

(showing the intermediate results using a graphic user interface (GUI)), and then save the generated map as an initial map. A group of data annotation experts then review the generated map and correct any errors. The corrections by the experts is then sent back to the machine learning module to improve detection of map elements. Finally, the corrected map is used with the input data to create the End-to-End data set, which is used for a machine learning step to generate a complete scene from the data.

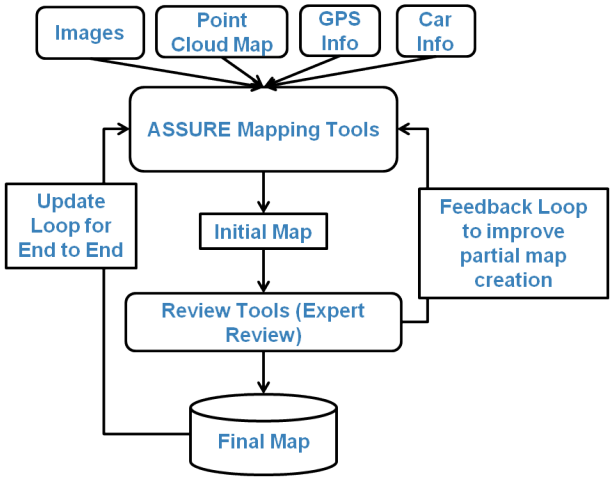


Figure 5-6: ASSURE Maps system design.

5.2.3 Experiment Results

We created LIDAR processing APIs to detect road lines, curbs and markings. We also created APIs to process video images to detect road lines and markings. We then fused these results to improve detection accuracy. We also created a Deep Learning module to detect traffic objects such as intersections, stoplights, road signs and markings, then projected the results to the LIDAR frame. Finally, we stitched together the results from all of the modules and used semantic road rules to extract the final road network map.

Figure 5-7 shows the extracted map items from the LiDAR data, showing we were able to successfully detect curbs, lines and markings. Figure 5-8 shows the extracted information from the vision detection system, which allowed us to detect traffic light,

intersections, markings and lines.

Figure 5-9 shows the resulting map after fusion of the data from the LiDAR and visual modules. The generated map is projected on the ground truth map to verify the results. By the end of the project, we were able to create an HD map of an entire road in 10 minutes, which previously took days to create manually.

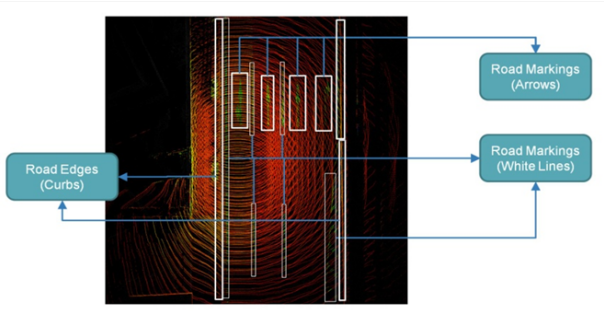


Figure 5-7: LIDAR based detection of the map data (curbs, markings and lines).

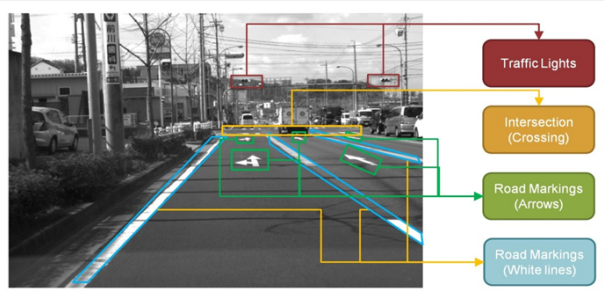


Figure 5-8: Visual detection of map data (traffic lights, intersections, markings and lines).

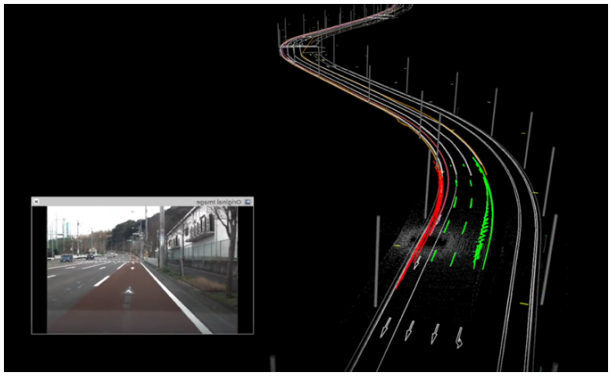


Figure 5-9: Resulting map after fusing data from LiDAR and visual detection modules.

5.2.4 RWDC in ASSURE maps

The relationship between ASSURE Maps and RWDC is two-fold. The first example is shown in Figure 5-10. The ASSURE Maps business model is based on using customer data - with customer approval of course - to improve our detection and mapping algorithms. The second example is the data circulation that occurs at the core of the system development process. Figure 5-11 shows how data is utilized within the ASSURE mapping system.

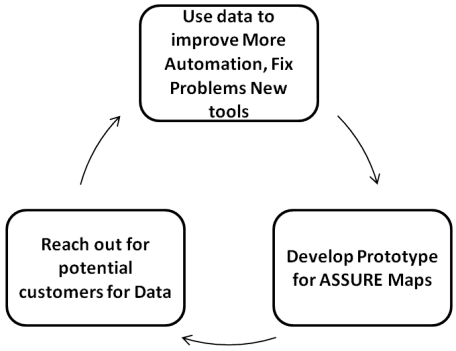


Figure 5-10: Relationship between ASSURE Maps and RWDC from a business perspective.

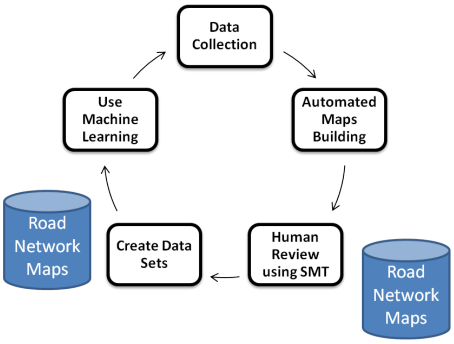


Figure 5-11: Relationship between ASSURE Maps and RWDC from a system development perspective.

5.3 Autonomous driving Tech-Lead Internship

One of the major requirements for graduation from the Leading Program (and also a great learning opportunity) is participation in Global Challenge II (GC II), which

requires participants to work internationally with a top university or company R&D department active in the same field as their field of study. The required work period is from three to six months, with the objective of enabling the student to experience working abroad in cooperation with an international team to achieve state-of-the-art research results. During my internship I worked with a UK-based technology company called Linaro for 3 months. It was a great opportunity to utilize my past experience, learn about new technology, meet new people, face new challenges and gain new experiences. My main motivation was to link my PhD studies with work in the autonomous driving technology industry.

5.3.1 Introduction

Linaro is an open-source collaborative engineering organization developing software for the **Arm** ecosystem. In 2010, they began a new initiative to build state-of-the-art autonomous vehicle software components and architecture, leveraging systems-on-a-chip (SoCs) from the Arm ecosystem for use with open-source autonomous driving frameworks such as Autoware. As an autonomous driving tech-lead, there were many tasks I needed to handle, which involved both software and hardware projects, as shown in Figure 5-12.

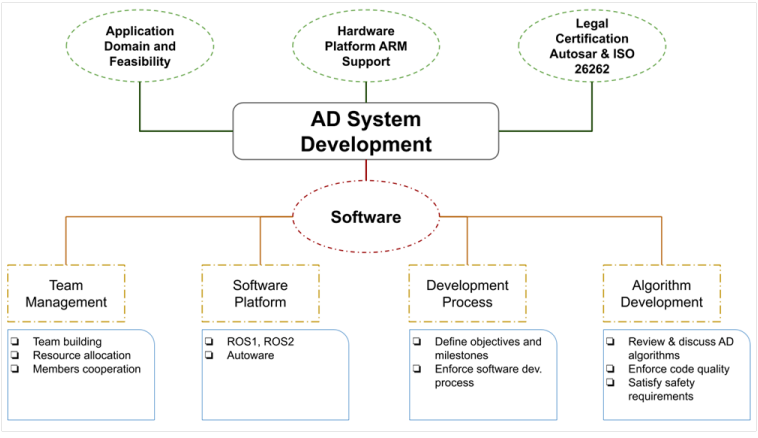


Figure 5-12: Roles of an Autonomous Driving Tech-Lead. They have many technical roles, but also have non-technical (business) tasks.

5.3.2 Autoware Performance analysis

Producing high performance SoCs for autonomous driving applications is essential, and part of my work was to evaluate the performance of the full Autoware autonomous driving software stack when operating on the following computing machines:

- A powerful, Intel-based desktop computer with GPU chip support.
- A mid-range, Intel-based laptop without GPU chip support.
- An SoC supported ARM developer box PC, developed through the collaborative work of multiple open architecture/open-source initiatives.

Figure 5-13 shows a comparison of the performance of the three computing platforms. The testing involved launching the full Autoware stack on each machine and logging the performance for each module separately. The Autoware modules tested were Localization, Object Detection, Planning, Visualization and Data Playback.

Platform	Memory	Processor	GPU	Additional Accelerator	Power
Powerful Intel Based Desktop	32 GB	Intel Zeon 3.6GHz	Nvidia 1080		500+ W
Medium Intel Based Laptop	16 GB	Intel i7 2.2 GH z	Build In Graphics Acceleration		40+ W
ARM Developer Box (SynQuacer)	4 GB	24 Cores Arm Cortex-A53 1.0 GHz	Standard Card	USB AI accelerator	25-30 W

Figure 5-13: Comparison of the three main computing machines used for Autoware performance testing.

Performance results are shown in Figure 5-14, which clearly shows that performance depends on the type of algorithm and implementation used. The powerful PC outperforms the other two computing machines with almost all of the modules. Only for image-based object detection did the developer box equipped with an AI accelerator outperform the other two computing machines. It is important to take advantage of all of the processing power provided by the SoC, and the developer box has 24 ARM processors at its disposal. In Figure 5-15 we show the results of a performance comparison when using a special localization implementation designed for

use with multiple processors. We can see that the ARM-based chip can outperform the powerful PC when a customized solution is introduced.

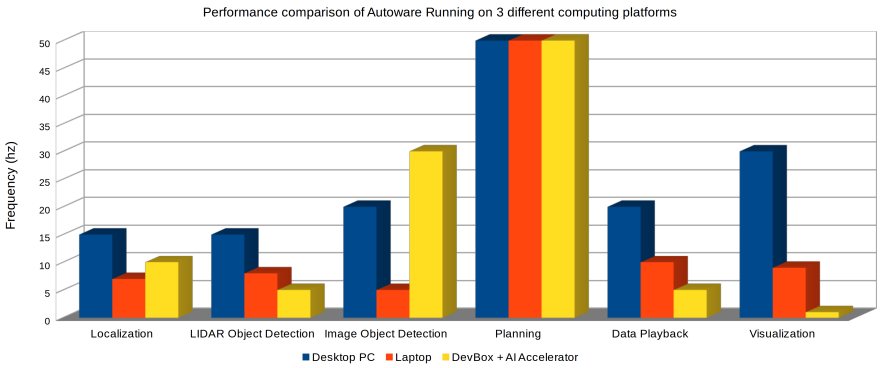


Figure 5-14: Comparing the performance of testing each Autoware module on each platform.

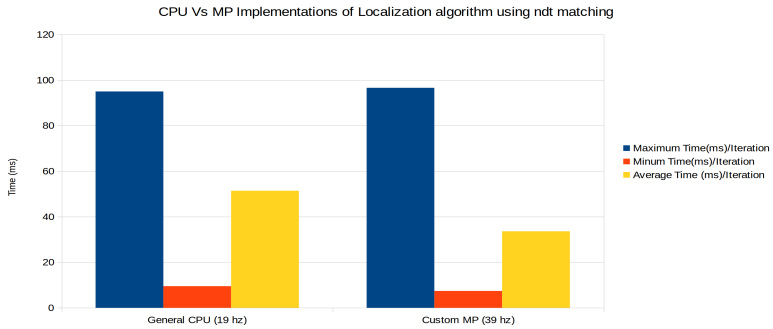


Figure 5-15: Comparison between two implementation of Localization, one is using general purpose processor, and the other is leveraging the multiprocessor capability of the computing machine.

Another task I was assigned during my internship was the creation of an internal report that summarized results for the following:

- Internal resources assessment
- Linaro road map overlapping the Autoware foundation road map
- Linaro's contribution to the Autoware foundation within the ARM eco-system
- Future designs needed to satisfy ISO 26262 safety standards
- ARM platform problems and proposed solutions

5.3.3 RWDC for the Internship

The work at Linaro establish RWDC in similar way to that of any of work on autonomous driving systems, as shown in Figure 5-16.

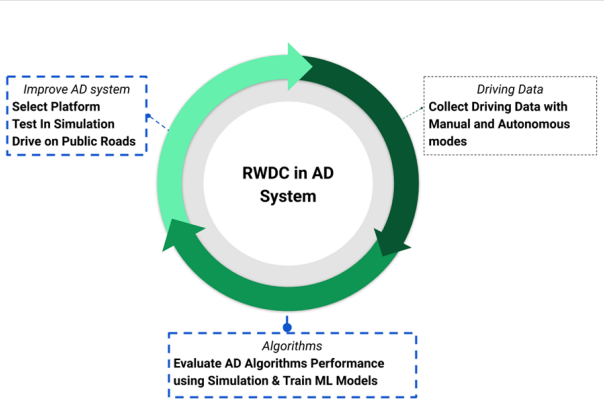


Figure 5-16: Relationship between RWDC and my work at Linaro for GC II.

By the end of my internship, I was able to understand autonomous driving systems from a different perspective, learn about new computing methods designed for automotive applications (SoCs and AI accelerators), and learn about the automotive functional safety requirements for autonomous driving systems (ISO 26262).

Chapter 6

Conclusion and Future Work

Autonomous driving is an important part of the effort to save lives by improving traffic safety. The development of better planning systems for autonomous driving is at the heart of efforts to improve autonomous driving, as the development of better planning techniques is essential for safer autonomous driving applications. Planning for autonomous driving is not straightforward, since many different modules need to be developed and integrated, and many different situations need to be anticipated and handled correctly. The main objective of a successful autonomous driving planner is generation of smooth, obstacle-free trajectories that can be followed by a trajectory-following controller. Although hundreds of autonomous vehicles are now operating on public streets every day, collecting data and testing new technologies, there are still many problems to be resolved with current state-of-the-art automated driving systems, such as portability, accuracy and maneuvering smoothly through complex intersections.

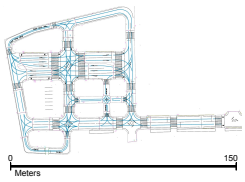
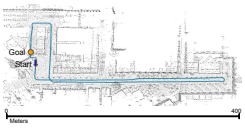
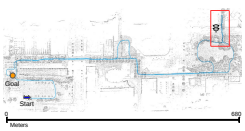
Autonomous driving planning modules such as global planning, local planning, intention and trajectory prediction, and behavior planning are essential, and planners should be able to handle dynamic environments, successfully generate routes using complex maps, estimate the intentions of surrounding vehicles and generate safe and smooth trajectories.

6.1 Conclusion

In this dissertation I have described my work to develop a complete, integrated, open-source planner for autonomous driving. The proposed implementation of this planner, **OpenPlanner**, is open-source, so the robotics community can freely take advantage of it, use it, modify it and build on it. OpenPlanner relies heavily on efficiently generated road network maps to improve driving safety.

In Chapter 3 I described the use of dynamic programming for global planning by searching the road network map using multiple additive costs. I also explained how the local planner generates trajectories, outlined behavior state generation using state machine transition and explained the functions and usage of related ROS nodes. I then described multiple experiments using both simulation and a modified, motorized scooter to demonstrate the effectiveness of the OpenPlanner integrated planner, by showing that it can operate effectively and safely in dynamic traffic environments. Table 6.1 shows summary of these experiments.

Table 6.1: Experimental results for proposed integrated planner.

Experiment	Environment	Results
Simulation		- The planner can correctly generate trajectories and select suitable behavior. - The global planner was able to find shortest path on a complex map.
Nagoya University		- The planner was able to perform in real time at a minimum of 10hz, with an average of 100 objected detected. - The planner was able to generate smooth trajectories and avoid obstacles even in narrow passage. - The planner was able to handle different traffic situations.
Tsukuba Challenge		- The planner achieved automatic rerouting even when perception-based behavior was not available. - The planner successfully navigated through a busy, unfamiliar environment.

In addition to our own experiments at Nagoya University, OpenPlanner has also been used by several other projects such as:

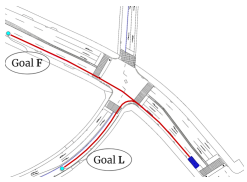
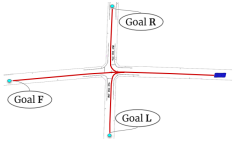
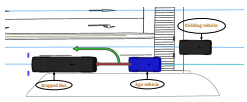
- Roboat project [22][23].

- Autonomous driving map editor [24].

- ADAS Demo in Hong Kong and USA.

In Chapter 4, I introduced an intention and trajectory estimation method for predicting the actions of surrounding vehicles by associating a probability with each intention and trajectory. This is a very important step before decision making in an autonomous driving system's planning process. A behavior planner is used to model the expected motion of surrounding vehicles, and particle filters are associated with each intention and trajectory. After that, the particle weights are calculated by finding the difference between the expected state and actual sensing information. Multiple sensing inputs, such as position, velocity, acceleration and turn signal status are used. Our results show that the proposed method has the ability to accurately discriminate between various possible intentions and trajectories in a variety of complex driving situations. A summary of our experimental results is shown in Table 6.2.

Table 6.2: Experimental results for intention and trajectory estimation.

Experiment	**Environment**	**Results**
Three way Intersection		- The estimator successfully assigned higher probabilities to the correct intention and trajectory for both Goal **F** and Goal **L**.
Four Way Intersection		- The method showed successful estimation of intention and trajectory when there was sufficient sensing information. - The method was tested in different situations, such as the vehicle stopping and not stopping at a stop line, and using or not using a turn signal. - There was only one situation in which the estimator could not predict the trajectory, which was when the vehicle stopped at the stop line and no turn signal information was available. However, even the best human driver can't predict where the vehicle will go next in this situation
Bus Stop (Parking & Yielding)		- Our method successfully estimated the parking intention of the bus and the intention of the vehicle in the other lane as to whether it would or would not yield. This is important because the planner needs to decide whether to wait behind the bus or to pass it.

6.2 Future Work

Although our integrated autonomous driving planner has been used successfully with different platforms in multiple environments, it still has not been tested on public roads. Achieving a speed of more than 30 kilometers/hour on a public road is one of our future goals. In addition, the intention and trajectory estimator is currently integrated chauvinistically with the planner. Another future goal is to use MDP to calculate optimal motion actions rather the current optimization method. It would also be interesting to try other methods of trajectory representation, such as representation using polynomials. Finally, for intention estimation filtering, using other probabilistic methods such as an HMM could improve performance dramatically.

List of Publications

Journal Publications

1. Open Source Integrated Planner for Autonomous Navigation in Highly Dynamic Environments, Journal of Robotics and Mechatronics (JRM) Vol.29 No.4, August 2017.

2. Estimating The Probabilities of Surrounding Vehicle's intentions and Trajectories using a Behavior Planner, International Journal of Automotive Engineering (IJAE), Vol.10 No.4, October 2019.

Appendix A

Implementation

In this section, we explain how the open source nature of OpenPlanner makes it easy for other engineers and researchers to use and modify it. OpenPlanner's source code is part of the Autoware framework and is divided into two parts, op global plnner and op local planner. Both set of nodes use three shared libraries, op_utility.so, op_planner.so and op_simu.so, which contain reusable functionality for all planning tasks. Since the OpenPlanner ROS nodes basically use these libraries as the system's core functionality, the basic planner could be used outside ROS with any other platform, for example as in [67]. [72].

A.0.1 op global planner node

The op global planner node functions as a global planner by finding the reference path to the goal position, and can be run upon request. The reference path contains multiple trajectories which guide the local planner to execute lane changes. By disabling the lane change option, only one trajectory is returned, if possible. Inputs for the global planner node include a vector map, current position and goal position. Output is the shortest path or paths from the map after taking into account predefined traffic costs, as shown in Figures 3-7 and 3-8 .

Table A.1: Global planner parameters description

Parameter	Description
pathDensity	distance between every two way points in the generated path. the longer the path the slower fixing the path density will be. recommended values ranging from [1,0.1] meter. it is useful when spars vector map is used.
enableSmoothing	use CG to smooth the generated path. only use this if the vector map has misaligned and spars way points.
enableLaneChange	when the vector map includes lane change information the planner will find multiple small trajectories which contains lane change information and one reference path to goal. when enabling the lane change option global planner should be called regularly to plan for the next lane change.
enableRvizInput	in simulation mode user can select goal position using rviz [ref to rviz], if not local planner will send the goal information using the proper ros topic name.
mapSource	select between different map sources, currently the node can use Autoware map messages or load the map from our custom .kml file
mapFileName	when mapSource is set to 2, this should contain the kml map file name

The launch file for the global planner node has several parameters that enable users to control the behavior and performance of the node, as shown in Table A.1 Additionally, the global planning node could be used with any local planner. It is the responsibility of the task planner to invoke the global planner node and send it the goal position. Currently, the goal position can also be specified using rviz (ROS visualization); in simulation mode, start and goal positions are specified in this manner.

A.0.2 op local planner nodes

The local planner nodes functions as a local planner. When there is a global path available, it generates roll-out trajectories and then selects the best one to output, depending on the obstacles detected. Also it outputs behavior state messages (current state, maximum velocity, minimum velocity, stopping distance, following distance, following velocity). Values in these messages are calculated to support a variety of different controllers. In our testing environment, we used a feed-forward PID controller which only uses current trajectory, state and maximum velocity. Other controllers may use following distance or velocity to generate smoother control signals. Users can choose to run off a way-point follower or use Pure Pursuit nodes from Autoware to follow the generated trajectory. Description of the important parameters for local

Table A.2: Local planner parameters description

Parameter	**Description**
mapSource	select between different map sources, currently the node can use Autoware map messages or load the map from our custom .kml file
mapFileName	when mapSource is set to 2, this should contain the kml map file name
maxVelocity	maximum velocity that planner should not exceed
maxLocalPlanDistance	length of the local trajectory roll-outs
samplingTipMargin	length of the roll-outs tip margin
samplingOutMargin	length of roll-outs roll in margin
rollOutDensity	distance between each two roll-out trajectories
rollOutsNumber	number of roll-outs not including the center trajectory, this number should be even number
pathDensity	distance between each way-points of the local trajectory
minFollowingDistance	distance threshold for exiting following behavior
maxFollowingDistance	distance threshold for entering following behavior
minDistanceToAvoid	distance threshold for obstacle avoidance behavior
enableSwerving	enable obstacle avoidance inside the lane (no lane change)
enableFollowing	enable following next car with the same velocity
enableTrafficLightBehavior	enable wait for traffic light to be green, if this is enabled and not traffic light detection is available, planner will assume that it is always red
enableLaneChange	enable obstacle avoidance to through lane change (over tack)
width	vehicle width in meters
length	vehicle length in meters
wheelBaseLength	vehicle wheel base in meters
turningRadius	vehicle min turning radius in meters

planner node is provided in Table A.2.

Appendix B

Algorithms

B.1 Algorithms

In this section we list the main algorithms used in OpenPlanner. The algorithm in Table B.1 fix errors in the vector map raw data by interpolating points along the path with fixed distance. In Tables B.2 and B.3 the main algorithm for finding global path between start and goal points. The algorithm in Tables B.4 and B.5 used to generate roll out trajectories for the local planner.

Table B.1: Algorithm for interpolating path points with fixed density.

Algorithm 1 Adjust path waypoints density (maxDistance)

```
01: remainingDistance = 0
02: for I = 0 to pathSize() - 1
03:     d := distance(Pi, Pi+1)
04:     nNewPoints := d / maxDistance
05:     if remainingDistance == 0
06:         newPath := Pi
07:     end if
08:     a = angle(Pi+1, Pi)
09:     for j = 0 to nNewPoints -1
10:         Pix' := Pix' + maxDistance * cos(a)
11:         Piy' := Piy' + maxDistance * sin(a)
12:         newPath := Pi
13:     end for
14: end for
15: return newPath
```

Table B.2: Algorithm for finding global path connecting start point to goal point.

Algorithm 2 FindGlobalPath (sNode , goal)

```
01: procedure FindGlobalPath(sPose, gPose)
02:     sNode := FindStartNode(sPose)
03:     gNode := BuildSearchTree(sNode, gPose)
04:     path := emptyPath
05:     paths := emptyPathsList
06:     TraceToStart(gNode, sNode, path, paths)
07:     return paths
08: end procedure
```

Table B.3: Detailed algorithms of procedure in Table B.2.

Algorithm 2 procedures

```
01: procedure BuildSearchTree
02:     s := sNode
03:     while (not s.isEmpty ) do
04:       node = popSmallestCost(s)
05:       if IsGoalAchieved(node, goal, MIN-DIST)
06:         return node
07:       end if
08:       if IsNewNode(node)
09:         if node.LeftLane
10:           synamic leftNode = ExpandLeft(node.LeftLane)
11:             CalculateCost(node, leftNode)
12:             s := leftNode
13:         end if
14:         if node.RighttLane
15:             rightNode = ExpandRight(node.RighttLane)
16:             CalculateCost(node, rightNode)
17:             s := rightNode
18:         end if
19:         for i = 0 to node.NextNodes
20:             nextNode = ExpandNext(node.NextNodes[i])
21:             CalculateCost(node, nextNode)
22:             s := nextNode
23:         end for
24:       end if
25:     end while
26: end procedure
27:
28: procedure TraceToStart(node, sNode, path, paths)
29:     if (node NOTEQUAL sNode)
30:       if (node.previousNodes.size() > 0)
31:         paths := path
32:         prevNode = FindMinCost(node.previousNodes)
33:         TraceToStart(prevNode, path, paths)
34:         node.dir = FORWARD
35:         path := node
36:       else if (node.leftNode)
37:         TraceToStart(node.leftNode, path, paths)
38:         node.dir = LEFT
39:         path := node
40:       else if (node.rightNode)
41:         TraceToStart(node.rightNode, path, paths)
42:         node.dir = RIGHT
43:         path := node
44:       end if
45:     end if
46: end procedure
```

Table B.4: Algorithm to generate roll out trajectories used in local planner.

Algorithm 3 GenerateRollOuts(sPose, maxDistance, globalPath, nRollOuts)

```
path-section := ExtractPlanningSection(sPose, maxDistance, globalPath)
SampleTrajectories(sPose, path-section, nRollOuts)
rollouts := SmoothRollOuts(rawRollOuts)
return rollouts
```

Table B.5: Detailed algorithms of procedure in Table B.4.

Algorithm 3 procedures

```
procedure ExtractPlanningSection(sPose, maxDistance, globalPath)
    index := FindClosestWaypoint(sPose, globalPath)
    while distance LESSTHAN maxDistance
        pathSection := globalPath(index++)
    end while
    FixPathDensity(pathSection)
    SmoothPath(pathSection)
    return pathSection
end procedure

procedure SampleTrajectories(sPose, pathSection, nRollOuts, carTipMargin, rollInMargin, rollOutMargin)
    rollOuts := CreateList(nRollOuts)
    for i = 0 to nRollOuts
        rollOuts[i] = AddCarTipSection(pathSection, carTipMargin)
        distanceFromCenter = rollOutDensity * (i - nRollOuts/2)
        rollOuts[i] = SampleRollInSection(pathSection, distanceFromCenter, rollInMargin)
        rollOuts[i] = AddRollOutSection(pathSection, rollOutMargin)
        SmoothPath(rollOuts[i])
    end for
    return rawRollOuts
end procedure

procedure SmoothRollOuts(rawRollOuts)
    return rollOuts
end procedure
```

Bibliography

[1] W. H. Organization, "Global status report on road safety," *Public Report*, 2018.

[2] W. H. Organization, "Death on the road." `https://extranet.who.int/roadsafety/death-on-the-roads/`, 2018. [Online; accessed December-2019].

[3] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA Grand Challenge: The Great Robot Race*. Springer Publishing Company, Incorporated, 1st ed., 2007.

[4] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Springer Publishing Company, Incorporated, 1st ed., 2009.

[5] U. D. of Transportation, "Automated driving systems 2.0, a vision for safety," *Department of Transportation website*, 2018.

[6] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka, "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions," *Transportation Research Part C: Emerging Technologies*, vol. 60, pp. 416–442, 2015.

[7] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[8] E. Marder-Eppstein, "Ros navigation stack." `http://wiki.ros.org/navigation/`, 2019. [Online; accessed December-2019].

[9] R. Design and C. L. . CIT, "Open-rdc." `https://github.com/open-rdc/`, 2019. [Online; accessed December-2019].

[10] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[11] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm.," in *ICAPS*, vol. 5, pp. 262–271, 2005.

[12] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[13] X. Li, Z. Sun, A. Kurt, and Q. Zhu, "A sampling-based local trajectory planner for autonomous driving along a reference path," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pp. 376–381, IEEE, 2014.

[14] S. Brechtel, T. Gindele, and R. Dillmann, "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, 2014.

[15] H. Liu and L. Wang, "Gesture recognition for human-robot collaboration: A review," *International Journal of Industrial Ergonomics*, vol. 68, pp. 355–367, 2018.

[16] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[17] C. Chen, A. Gaschler, M. Rickert, and A. Knoll, "Task planning for highly automated driving," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, pp. 940–945, IEEE, 2015.

[18] N. University, "Autoware foundation gitlab link." `https://gitlab.com/autowarefoundation/autoware.ai/`, 2019. [Online; accessed December-2019].

[19] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[20] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 287–296, IEEE, 2018.

[21] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[22] W. Wang, B. Gheneti, L. Mateos, F. Duarte, C. Ratti, and D. Rus, "Roboat: An autonomous surface vehicle for urban waterways," *IEEE Robotics and Automation Letters (submitted)*, 2019.

[23] M. . AMS, "Roboat project." `https://roboat.org`, 2019. [Online; accessed December-2019].

[24] W. N. Tun, S. Kim, J.-W. Lee, and H. Darweesh, "Open-source tool of vector map for path planning in autoware autonomous driving software," in *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pp. 1–3, IEEE, 2019.

[25] H. Darweesh, E. Takeuchi, K. Takeda, Y. Ninomiya, A. Sujiwo, L. Y. Morales, N. Akai, T. Tomizawa, and S. Kato, "Open source integrated planner for autonomous navigation in highly dynamic environments," *Journal of Robotics and Mechatronics*, vol. 29, no. 4, pp. 668–684, 2017.

[26] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "The interactive museum tour-guide robot," in *Aaai/iaai*, pp. 11–18, 1998.

[27] Y. Morales, A. Carballo, E. Takeuchi, A. Aburadani, and T. Tsubouchi, "Autonomous robot navigation in outdoor cluttered pedestrian walkways," *Journal of Field Robotics*, vol. 26, no. 8, pp. 609–635, 2009.

[28] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard, "Autonomous robot navigation in highly populated pedestrian zones," *Journal of Field Robotics*, vol. 32, no. 4, pp. 565–589, 2015.

[29] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

[30] S. M. LaValle, "Motion planning," *IEEE Robotics Automation Magazine*, vol. 18, no. 2, pp. 108–118, 2011.

[31] S. M. LaValle, "Motion planning: The essentials," *IEEE Robotics Automation Magazine*, vol. 18, no. 1, pp. 79–89, 2011.

[32] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[33] S. Koenig and M. Likhachev, "D^* lite," *Aaai/iaai*, vol. 15, 2002.

[34] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.

[35] P. Beeson, N. K. Jong, and B. Kuipers, "Towards autonomous topological place detection using the extended voronoi graph," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 4373–4379, IEEE, 2005.

[36] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, *et al.*, "Junior: The stanford entry in the urban challenge," *Journal of field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[37] P. Czerwionka, M. Wang, and F. Wiesel, "Optimized route network graph as map reference for autonomous cars operating on german autobahn," in *The 5th International Conference on Automation, Robotics and Applications*, pp. 78–83, IEEE, 2011.

[38] R. Simmons, "The curvature-velocity method for local obstacle avoidance," in *Proceedings of IEEE international conference on robotics and automation*, vol. 4, pp. 3375–3382, IEEE, 1996.

[39] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[40] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, vol. 1, pp. 341–346, IEEE, 1999.

[41] J. J. Park and B. Kuipers, "A smooth control law for graceful motion of differential wheeled mobile robots in 2d environment," in *2011 IEEE International Conference on Robotics and Automation*, pp. 4896–4902, IEEE, 2011.

[42] Y. Morales, A. Watanabe, F. Ferreri, J. Even, T. Ikeda, K. Shinozawa, T. Miyashita, and N. Hagita, "Including human factors for planning comfortable paths," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6153–6159, IEEE, 2015.

[43] R. Ding and W.-H. Chen, "Moving object tracking in support of unmanned vehicle opeartion," in *2013 19th International Conference on Automation and Computing*, pp. 1–6, IEEE, 2013.

[44] R. Pepy, A. Lambert, and H. Mounier, "Reducing navigation errors by planning with realistic vehicle model," in *2006 IEEE Intelligent Vehicles Symposium*, pp. 300–307, IEEE, 2006.

[45] S. Ammoun and F. Nashashibi, "Real time trajectory prediction for collision risk estimation between vehicles," in *2009 IEEE 5th International Conference on Intelligent Computer Communication and Processing*, pp. 417–422, IEEE, 2009.

[46] G. Xie, H. Gao, L. Qian, B. Huang, K. Li, and J. Wang, "Vehicle trajectory prediction by integrating physics-and maneuver-based approaches using interactive multiple models," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 7, pp. 5999–6008, 2017.

[47] A. Houenou, P. Bonnifait, V. Cherfaoui, and W. Yao, "Vehicle trajectory prediction based on motion model and maneuver recognition," in *2013 IEEE/RSJ international conference on intelligent robots and systems*, pp. 4363–4369, IEEE, 2013.

[48] D. Lenz, F. Diehl, M. T. Le, and A. Knoll, "Deep neural networks for markovian interactive scene prediction in highway scenarios," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 685–692, IEEE, 2017.

[49] J. Morton, T. A. Wheeler, and M. J. Kochenderfer, "Analysis of recurrent neural networks for probabilistic modeling of driver behavior," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1289–1298, 2016.

[50] D. J. Phillips, T. A. Wheeler, and M. J. Kochenderfer, "Generalizable intention prediction of human drivers at intersections," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1665–1670, IEEE, 2017.

[51] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[52] P. Liu, A. Kurt, *et al.*, "Trajectory prediction of a lane changing vehicle based on driver behavior estimation and classification," in *17th international IEEE conference on intelligent transportation systems (ITSC)*, pp. 942–947, IEEE, 2014.

[53] Z. Ghahramani, "An introduction to hidden markov models and bayesian networks," in *Hidden Markov models: applications in computer vision*, pp. 9–41, World Scientific, 2001.

[54] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee, "Intention-aware online pomdp planning for autonomous driving in a crowd," in *2015 ieee international conference on robotics and automation (icra)*, pp. 454–460, IEEE, 2015.

[55] K. Lidstrom and T. Larsson, "Model-based estimation of driver intentions using particle filtering," in *2008 11th International IEEE Conference on Intelligent Transportation Systems*, pp. 1177–1182, IEEE, 2008.

[56] C. Shen, A. Van den Hengel, and A. Dick, "Probabilistic multiple cue integration for particle filter based tracking," *Australian Pattern Recognition Society*, 2003.

[57] S. Vacek, S. Bergmann, U. Mohr, and R. Dillmann, "Rule-based tracking of multiple lanes using particle filters," in *2006 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pp. 203–208, IEEE, 2006.

[58] C. Hubmann, M. Becker, D. Althoff, D. Lenz, and C. Stiller, "Decision making for autonomous driving considering interaction and uncertain prediction of surrounding vehicles," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1671–1678, IEEE, 2017.

[59] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo em motion planner," *arXiv preprint arXiv:1807.08048*, 2018.

[60] VIRES, "OpenDRIVE Project." `http://www.opendrive.org/`, 2019. [Online; accessed December-2019].

[61] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[62] F. Poggenhans, J.-H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr, "Lanelet2: A high-definition map framework for the future of automated driving," in *Proc. IEEE Intell. Trans. Syst. Conf.*, (Hawaii, USA), November 2018.

[63] Y. Saad, *Iterative methods for sparse linear systems*, vol. 82. siam, 2003.

[64] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," tech. rep., Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.

[65] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *Ann Arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.

[66] T. City, "Tsukuba challenge." `http://www.tsukubachallenge.jp`, 2019. [Online; accessed December-2019].

[67] S. Muramatsu, T. Tomizawa, and S. Kudoh, "Development of intelligent mobile cart in a crowded environment: Robust localization technique with unknown objects (special issue on real world robot challenge in tsukuba: Technology for the coexistence with human beings)," *Journal of robotics and mechatronics*, vol. 26, no. 2, pp. 204–213, 2014.

[68] H. Darweesh, "Openplanner youtube channel." `https://www.youtube.com/playlist?list=PLVAImlqqGbr5G1EjscMom6_JcwjxCMX2g`, 2019. [Online; accessed December-2019].

[69] A. Koenig, T. Rehder, and S. Hohmann, "Exact inference and learning in hybrid bayesian networks for lane change intention classification," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1535–1540, IEEE, 2017.

[70] NHTSA, "Pre-crash scenario typology for crash avoidance research," *Department of Transportation website*, 2007.

[71] N. University, "Real World Data Circulation Program." `http://www.rwdc.is.nagoya-u.ac.jp/index-e.php`, 2019. [Online; accessed December-2019].

[72] H. Bruyninckx, "Open robot control software: the orocos project," in *Proceedings 2001 ICRA. IEEE international conference on robotics and automation (Cat. No. 01CH37164)*, vol. 3, pp. 2523–2528, IEEE, 2001.