Paper:

Open Source Integrated Planner for Autonomous Navigation in Highly Dynamic Environments

Hatem Darweesh*, Eijiro Takeuchi*, Kazuya Takeda*, Yoshiki Ninomiya*, Adi Sujiwo*, Luis Yoichi Morales*, Naoki Akai*, Tetsuo Tomizawa, and Shinpei Kato*****

*Nagoya University
Furo-cho, Chikusa-ku, Nagoya, Aichi 464-8601, Japan
E-mail: {hatem.darweesh@g.sp.m.is, e_takeuchi@i, ninomiya@coi, morales_yoichi@coi, akai@coi}.nagoya-u.ac.jp, kazuya.takeda@nagoya-u.jp, sujiwo@ertl.jp
**National Defense Academy of Japan
1-10-20 Hashirimizu, Yokosuka, Kanagawa 239-8686, Japan
E-mail: tomizawa@nda.ac.jp
***The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8654, Japan
E-mail: shinpei@pf.is.s.u-tokyo.ac.jp
[Received February 28, 2017; accepted June 9, 2017]

Planning is one of the cornerstones of autonomous robot navigation. In this paper we introduce an open source planner called "OpenPlanner" for mobile robot navigation, composed of a global path planner, a behavior state generator and a local planner. OpenPlanner requires a map and a goal position to compute a global path and execute it while avoiding obstacles. It can also trigger behaviors, such as stopping at traffic lights. The global planner generates smooth, global paths to be used as a reference, after considering traffic costs annotated in the map. The local planner generates smooth, obstacle-free local trajectories which are used by a trajectory tracker to achieve low level control. The behavior state generator handles situations such as path tracking, object following, obstacle avoidance, emergency stopping, stopping at stop signs and traffic light negotiation. OpenPlanner is evaluated in simulation and field experimentation using a non-holonomic Ackerman steering-based mobile robot. Results from simulation and field experimentation indicate that OpenPlanner can generate global and local paths dynamically, navigate smoothly through a highly dynamic environments and operate reliably in real time. OpenPlanner has been implemented in the Autoware open source autonomous driving framework's Robot Operating System (ROS).

Keywords: autonomous driving, path planning, open source software

1. Introduction

Autonomous robot navigation requires perception, localization, control and planning. Although there are currently many open source resources available to researchers for perception, localization and control, it is hard to find an open source planner that is general enough to be used directly or which can be easily modified to suite a particular application, because planning is the core module that connects everything together and it is also application dependent.

In this study we concentrate on two types of planning, path planning and behavior planning, since both are needed in completely autonomous navigation systems for mobile robots. Museum tour guide robots are one example of autonomous navigation in indoor robots [1]. Studies on long-range outdoor robot navigation [2, 3] have shown that state-of-the-art 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 [4, 5]. Even though there have been successful implementations of autonomous vehicles, autonomous navigation is still a difficult problem given the vast number of possible conditions in dynamic environments. As a result, there are few open source planners that can deliver reliable results. Therefore, our goal in this study was to develop an open source autonomous navigation system that can handle dynamic environments while being extensible enough for the open source community to use, customize and enhance.

The architecture of OpenPlanner is illustrated in **Fig. 1**. It includes a global planner that generates a global reference path from a vector (road network) map. The system then uses this global path to generate an obstacle-free local trajectory from a set of sampled roll-outs. At the center of the planner, the behavior generator uses predefined traffic rules and sensor data to function as an orchestrator, using collision and traffic rule cost calculations, selection of optimum trajectories, replanning commands and velocity profiling.

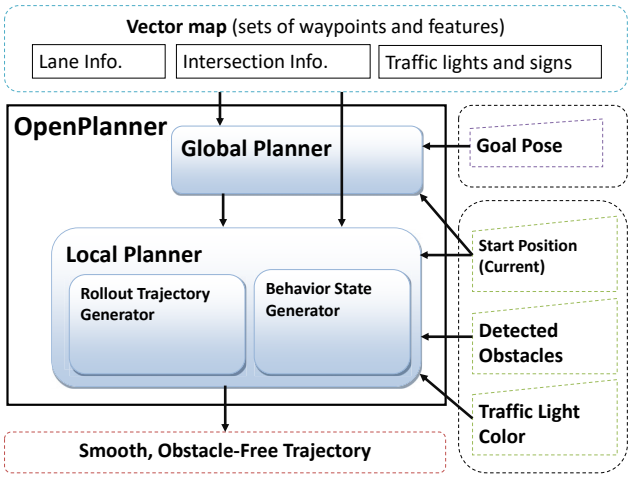


Fig. 1. OpenPlanner architecture.

The research presented in this paper was originally developed for autonomous driving applications, and the system can also be found as part of the Autoware framework [a]. Autoware is an open source autonomous driving framework developed by Nagoya University which is used by many researchers for autonomous driving research and development [6]. Autoware is based on the Robot Operating System (ROS) described in [7]. It is a collection of ROS packages such as OpenPlanner, plus additional helper libraries. OpenPlanner is general enough to work with any mobile robot by simply adjusting its parameters, and has been used with both differential drive and non-holonomic robots. In this study we use an Ackerman-based steering robot, based on a mobility scooter, which was used in the Tsukuba Real World Robot Challenge (RWRC) [b] 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 much 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.

In Section 2, we survey related state-of-the-art work. In Section 3, we provide an overview of the OpenPlanner system and its architecture. In Section 4, we give a detailed explanation of the system's global path planning method, including the use and structure of vector maps. The local planner is discussed in Section 5, and behavior state generation and the design of the state machine are explained in Section 6. In Section 7, our experimental set-up is described and our experimental results are evaluated quantitatively and qualitatively. We present our conclusions in Section 8. In Appendix A, we discuss current implementation and the wide range of platforms OpenPlanner can be used with. We also provide tips for users and developers for extension and parameter tuning. Algorithms are listed in Appendix B.

2. Related Work

In robotics, path planning is the task of finding a collision-free trajectory from a starting position to a goal location, and determines navigation decisions. Path planning has been widely studied, from simple collision avoidance in simulated environments [8] to advanced algorithms which include vehicle constraints and uncertainty [9, 10]. Path 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 [11]. 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 cite examples of related work for each specific aspect of our proposed planning system.

2.1. Global Planning

Some global path planners, such as A* [12] use heuristic functions while others, such as Anytime dynamic A* [13] and the D* algorithm [14] also employ replanning. There are topological approaches as well, such as Voronoi graphs [15] which compute collision free paths. These techniques are based on grid maps updated with sensor information, called cost maps. Such techniques create global planes for unstructured environments like off-road navigation and parking situations. Another type of global planning environment is a network of structured roads extending for several kilometers. With maps of this size, cost maps become impractical and different environment presentation method is needed.

At the 2007 DARPA Urban Challenge, the teams received a road network definition file (RNDF) for the complete course. The teams used this file to globally plan their motion through the challenge objectives, as described in [4, 5]. Dynamic programming techniques for global planning were used by Stanford University's team as described in [16], which involved dynamic programming with accelerating nodes.

Clearly, this kind of structured environmental information made it practical for teams to plan global paths including lane changes, intersection negotiation, stop signs, traffic lights and parking. Since then, RNDF has become essential for autonomous navigation. In [17], optimized

RNDF for autonomous driving was introduced and many companies are working on accurate RNDF-based maps called vector maps. Although the structure of these maps is similar to open street maps [8], they are much more precise and include additional, regularly updated, information.

2.2. Behavior State Machine

The other important planning function besides path planning is task planning, also called behavior generation, which generally uses a state machine to represent tasks and apply the rules that govern transitions between these tasks. In [18] 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 conditions in a symbolic space.

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 [14]. Other successful obstacle avoidance algorithms consider vehicle constraints while predicting the future position of the vehicle [19]. The global dynamic window approach integrates global path information and uses it for obstacle avoidance [20, 21]. More recently developed planners take human factors into account, in order to compute paths which are comfortable for passengers [16, 22]. Efficient methods of local planning which generate multiple roll-outs, starting from the center of a vehicle and running parallel to a reference path, have also been introduced [4, 23]. These roll-outs are then linearly sampled and optimized to satisfy vehicle kinematics.

Object tracking is an important component of many local planning approaches, including ours. Noisy sensors, faulty detection algorithms and weather conditions contribute to false positive and false negative detections. It is essential that local planning methods have reliable object tracking capabilities, especially in outdoor autonomous navigation applications. In [24], multiple hypothesis tracking (MHT) was used to achieve multiple target tracking, while other researchers have used probabilistic filters, such as Kalman or Particle filters.

2.4. Open Source

The two major open source planners currently available are the Open Motion Planning Library (OMPL) [25] and Navigation Stack [c]. OMPL is basically a collection of APIs which can be used with or without ROS, while Navigation Stack, on the other hand, is part of ROS and cannot be used outside it. 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 could be used directly with Autoware or any other ROS-based framework.

Open-rdc is an open source, robotic navigation programming software [d] based on the ROS Navigation Stack [c]. It can be used as an extension to Navigation Stack for applications involving differential-drive mobile robots. Open-rdc was developed for use at the Tsukuba RWRC in 2015.

OpenPlanner is intended to be used for autonomous navigation of mobile robots in general, and more specifically, for autonomous driving applications. It is designed to use vector maps or road network maps and use all the discrete information they contain, such as locations of traffic lights, traffic signs, intersections, stop lines, and so on, which is one of its main advantages over other 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 equation and using vector maps to handle that problem. Of course free space global planners like RRT* [26] and Hybrid A* are important for parking and off-road situations. 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 OpenPlanner has over Navigation Stack is that OpenPlanner can also be used with non-holonomic platforms.

The function of behavior state machines is hard to generalize because their use 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 the planner, 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, it requires only a vector map and goal location for global planning, and for local planning and behavior state generation it only requires current position and detected obstacles.

3. System Overview

Figure 1 shows the general architecture for OpenPlanner, the main three components of which are a global planner, a behavior state generator and a local planner.

3.1. Overview of Global Planner

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 [4]. 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 **Fig. 2**. We annotated the map manually with traffic rules and features, such as traffic lights and stop lines, from start to goal. More details will be provided in Section 4.

Fig. 2. Tsukuba Real World Robot Challenge vector map.

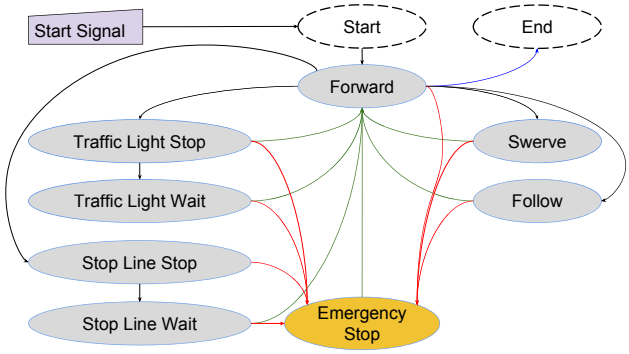


Fig. 3. Current system behavior states.

3.2. Overview of Behavior State Generation

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. **Fig. 3** shows the currently available states in the OpenPlanner system. More details are provided in Section 6.

3.3. Overview of Trajectory Generation

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. **Fig. 4** shows seven possible rollout trajectories, including the center one. We used a modified version of the Stanford approach presented in [4]. The implementation algorithms will be explained in Section 5.

4. Global Planner

In Section 2, we discussed the different approaches to path planning and the uses of each approach. OpenPlanner uses a vector map as the main input for global planning, and for the calculated reference path used by the local planner to generate roll-out trajectories. In this section, we elaborate further on vector maps and global planning.

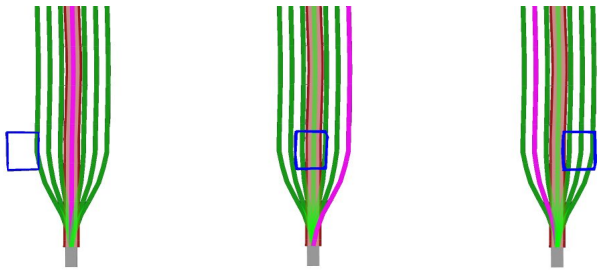


Fig. 4. Local planner in action, in (a) central trajectory is free, in (b) obstacle blocks central trajectory so the most feasible one was the most right trajectory, in (c) the most feasible one was the second on the left.

Table 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 waypoints	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

4.1. Vector Maps

One of the most widely used approaches for autonomous vehicle navigation is the use of vector maps, sometimes called high definition maps to differentiate them from maps used in geographic information system (GIS) applications such as open street maps [8]. Vector maps include data that autonomous navigation modules need to make sense of the surrounding environment. **Table 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 waypoints 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 density of lane center lines, as shown in **Table 6** 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 [27] is applied.

Although the Tsukuba RWRC map is very simple (**Fig. 2**), as is our testing map for the Nagoya University campus (**Fig. 5**), OpenPlanner also supports road network compatible vector maps such as the one shown in **Fig. 6**.

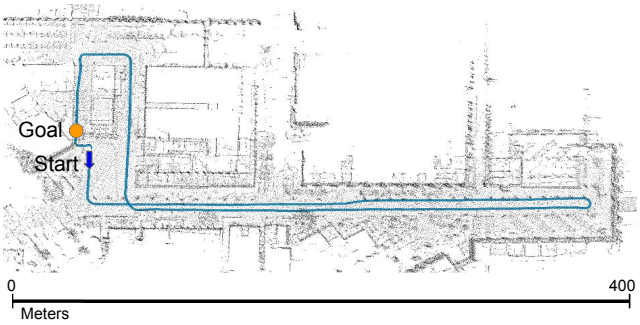


Fig. 5. Experimental vector map for Nagoya University.

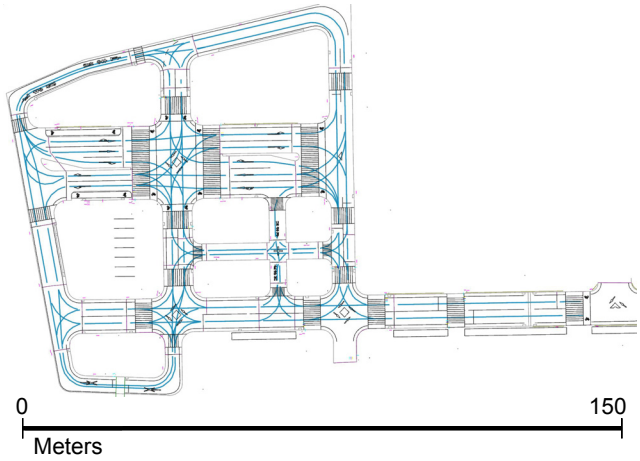
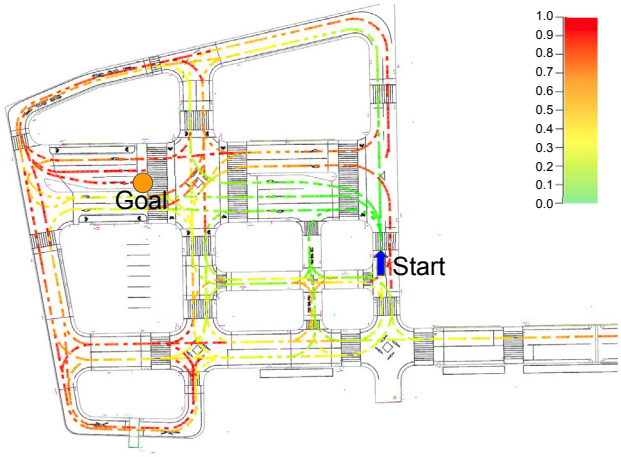
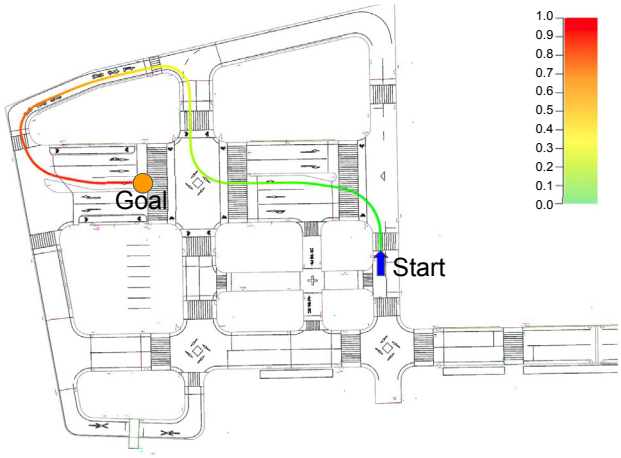


Fig. 6. Vector map with complex structures.



(a)



(b)

Fig. 7. Searching the map for the shortest path. Line color indicates route cost.

4.2. Global Planner

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 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 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 **Fig. 7(a)**. 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 **Fig. 7(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 **Figs. 8** and **9**, the latter of which includes a lane change. In **Fig. 5**, we show the global path for one of the testing environments, which simply represents the same vector map shown in **Fig. 5**. **Tables 7** and **8** in Appendix B show the algorithms used to find the global path.

5. Local Planner

A local trajectory planner is a set of functionalities that generate a smooth trajectory which can be tracked by path-following algorithms, such as Pure Pursuit [28]. For OpenPlanner, we adapted the roll-out generation approach (**Fig. 4**), 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.

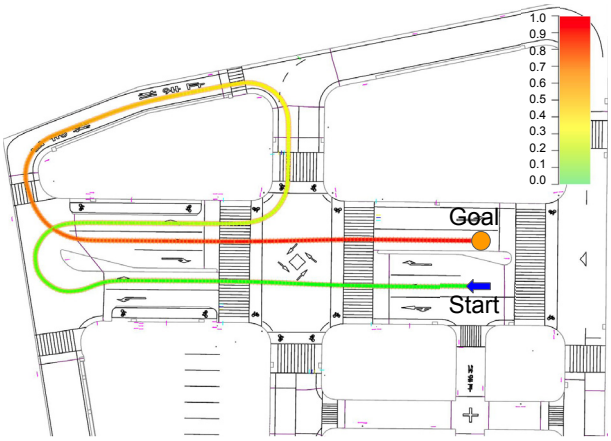


Fig. 8. Example of complex global planning.

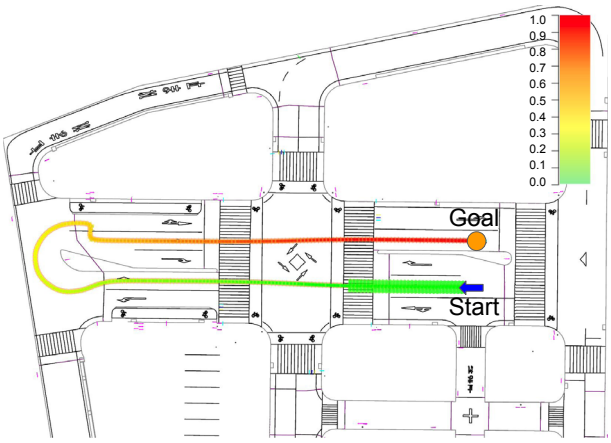


Fig. 9. Global planning including lane change.

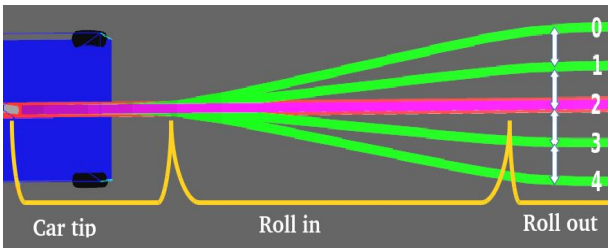


Fig. 10. Sections for generating roll outs.

5.1. Roll-Out 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 **Fig. 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.

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 6** 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) [29]. 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. **Fig. 11** shows the steps of roll-out generation, and implementation is demonstrated in **Table 9** in Appendix B.

5.2. Cost Calculation

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 9** 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

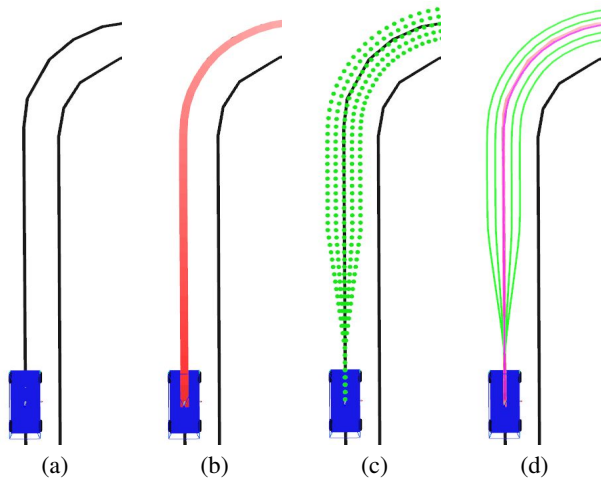


Fig. 11. Steps for generating local trajectories: (a) original map, (b) path section extracted from global path, (c) sampling, (d) smoothing using conjugate gradient.

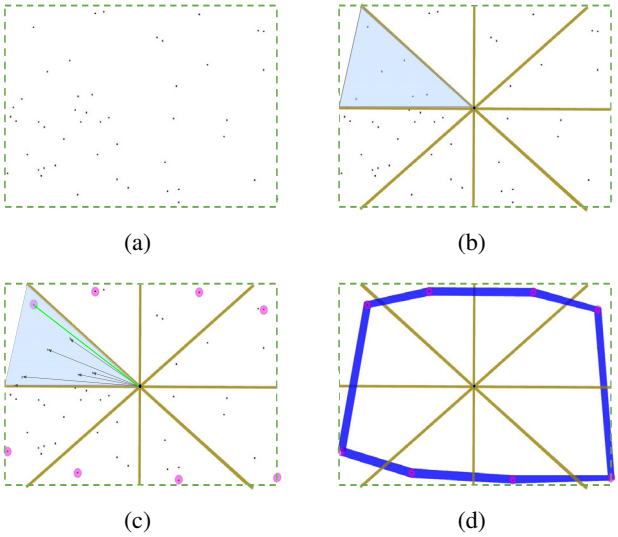


Fig. 12. Obstacle data representation using only the contour points from the point cloud data. In (a) we show sample random point cloud points. Step 1 of the contour calculation is shown in (b); we find the point cloud center point and from that point we divide the point cloud into 8 quarters, number 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.

Autoware [a] which outputs two types of obstacle representations, bounding boxes and clusters of point cloud data. 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.

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. **Fig. 12** shows an example of obstacle detection using 8 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.

5.2.1. 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.

5.2.2. 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.

5.2.3. 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. **Fig. 13** shows color coded center costs, and **Fig. 14** illustrates the normalized total cost when there is an obstacle.

6. Behavior Generation Using State Machine

Situations like stopping at a traffic light, deciding to change lanes, stopping and waiting at a stop sign and

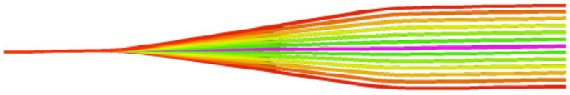


Fig. 13. Center cost keeps the robot in the center of a lane. Selected trajectory is at the center, with other trajectories gradient color represents cost.

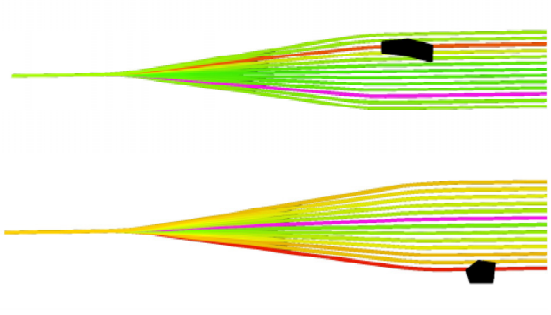


Fig. 14. Effect of an obstacle on cost calculation.

Table 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

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. **Fig. 3** shows the behavior states used for OpenPlanner during the Tsukuba Real World Robot Challenge. **Table 2** shows the transition rules for each state.

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.

Fig. 15. Experimental vector map with stop lines and traffic light.

7. Experimental Setup and Results

The first field test was conducted on the campus of Nagoya University, and a diagram of the route is shown in **Fig. 15**. The second field test took place at the Tsukuba RWRC event, the vector map of which is shown in **Fig. 2**. 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 **Figs. 6** and **7**. For field experiments, we added traffic information to the map manually.

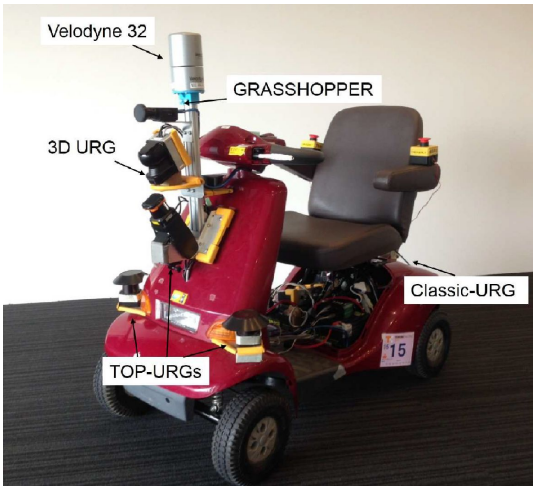


Fig. 16. Tsukuba Challenge hardware platform.

Table 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.

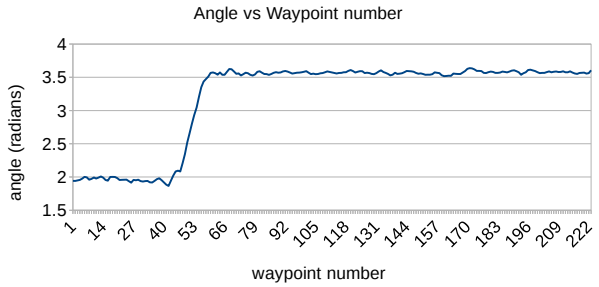
The experiment platform is shown in **Fig. 16**, same platform is used in [30]. It is a modified mobility scooter so it could be controlled by computer. It includes one HDL32 Velodyne LIDAR sensor which is used for localization and object detection. In addition to the 3D LIDAR we use three 2D LIDAR for 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.

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

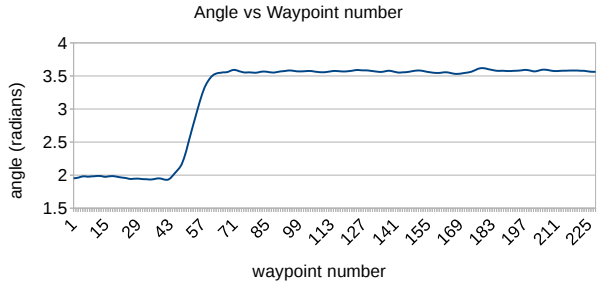
The **Table 3** shows the used parameter configurations for simulation and field experiments.

7.1. Qualitative Results

The first qualitative aspect of our experiment is to evaluate stability, which means OpenPlanner should work all



(a)



(b)

Fig. 17. The angle of each way-point shows curvature of the generated path with and without smoothing.

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 **Fig. 15**, the resulting sequence of driving behaviors were: Forward, Stop Sign, Wait Sign, Forward, Light Stop, Wait Light, Forward, Finish.

7.2. University Campus Experiment

Here we will concentrate on one section of the campus field test map, shown in **Fig. 15**, which consists of two straight lines, one curve, one stop line and one traffic light. **Fig. 17(a)** shows the curvature of the generated path, and the smoothed output of that path is shown in **Fig. 17(b)**. Smoothing improves path following performance without the need to relax the control parameters.

Results when navigating the map section shown in **Fig. 15** are displayed in **Fig. 18**, which shows the simulation results when navigating this section without any obstacles. Behaviors are represented by the blue line and be-

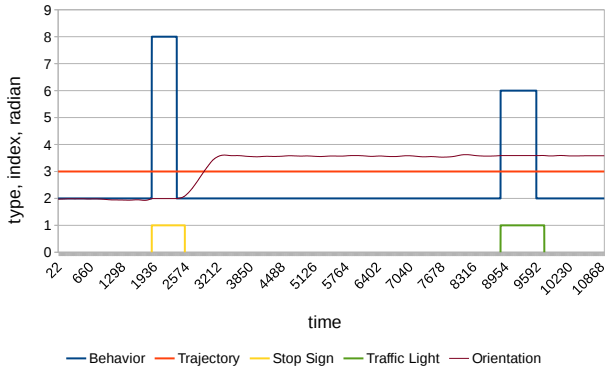


Fig. 18. Simulation test results when navigating without obstacles. Behavior ID values: 2 = move forward; 5 = stop for traffic light; 6 = wait for traffic light; 7 = stop for stop sign; 8 = wait for stop sign; 11 = avoid obstacle.

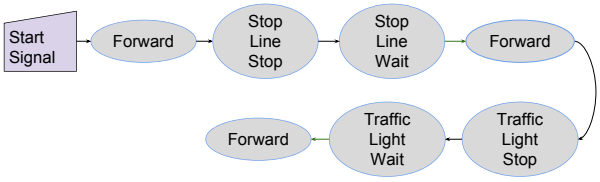


Fig. 19. Behavior state flow while navigating the simulated vector map without obstacles.

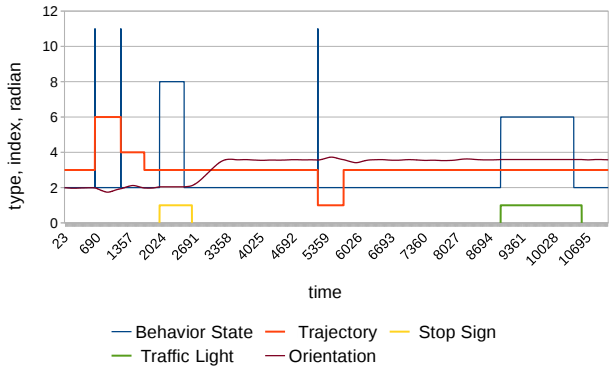


Fig. 20. Simulation test results when navigating with obstacles. Behavior ID values: 2 = move forward; 5 = stop for traffic light; 6 = wait for traffic light; 7 = stop for stop sign; 8 = wait for stop sign; 11 = avoid obstacle.

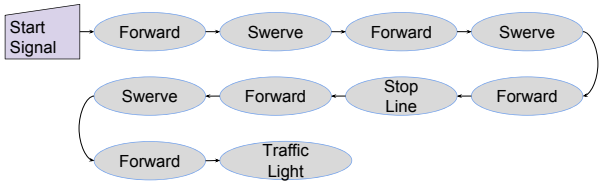


Fig. 21. Behavior state flow while navigating the simulated vector map with obstacles.

havior 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. **Fig. 19** illustrates the behavior transition flow during this experiment.

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. **Fig. 20** shows that both state transition and trajectory switching results were smooth. **Fig. 21** is a diagram of the behavior state transition flow when navigating this section with obstacles.

Field experiment results for the same map section, shown in **Fig. 22**, 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.

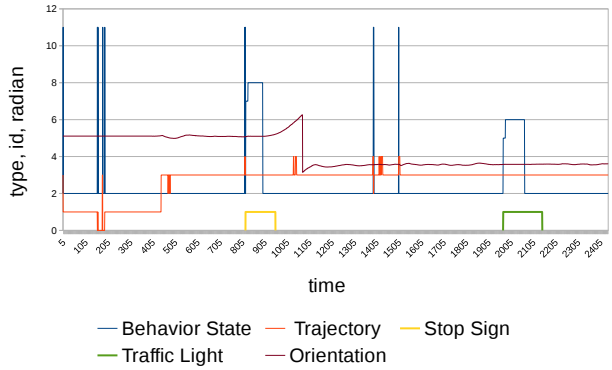


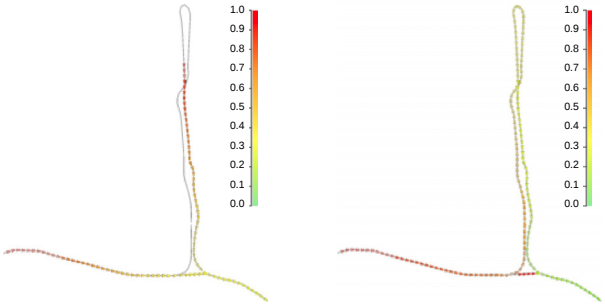
Fig. 22. Field test results. Behavior ID values: 2 = move forward; 5 = stop for traffic light; 6 = wait for traffic light; 7 = stop for stop sign; 8 = wait for stop sign; 11 = avoid obstacle.

7.3. 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 **Fig. 23(a)**. If we wanted to attempt the task and cross the street, we could simply increase the cost associated with the shortcut. **Fig. 23(b)** shows the planner choosing the longer route to avoid the high cost assigned to the shortcut. The resulting global paths are shown in **Figs. 24(a)** and **(b)** respectively. Behavior states when following both routes are simulated in **Figs. 25(a)** and **(b)**.

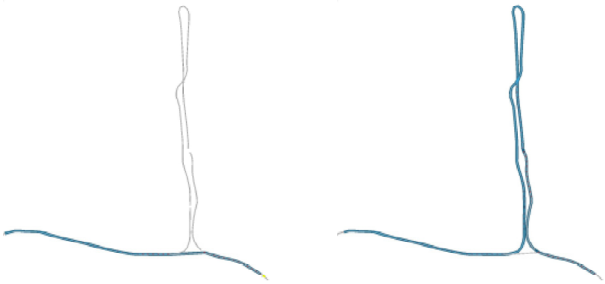
7.4. Performance

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 de-



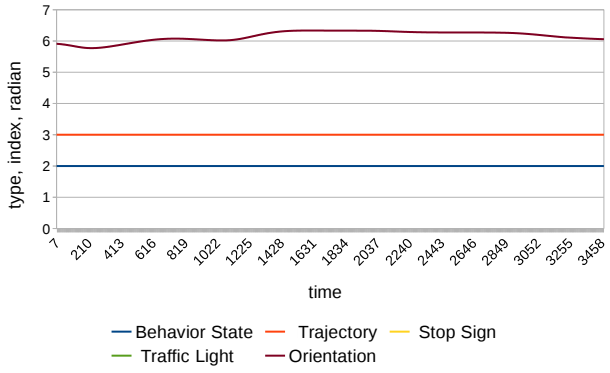
(a) (b)

Fig. 23. Section from Tsukuba Challenge path represented by rectangle in **Fig. 2**. Global planning dynamic cost calculation, (a) without and (b) with a high shortcut penalty.

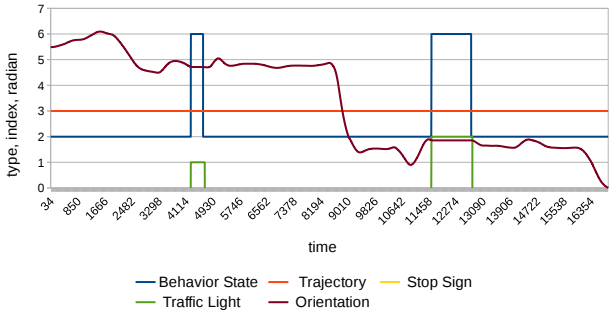


(a) (b)

Fig. 24. Section from Tsukuba Challenge path represented by rectangle in **Fig. 2**. Global planning final paths, (a) without and (b) with a high shortcut penalty.



(a)



(b)

Fig. 25. 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.

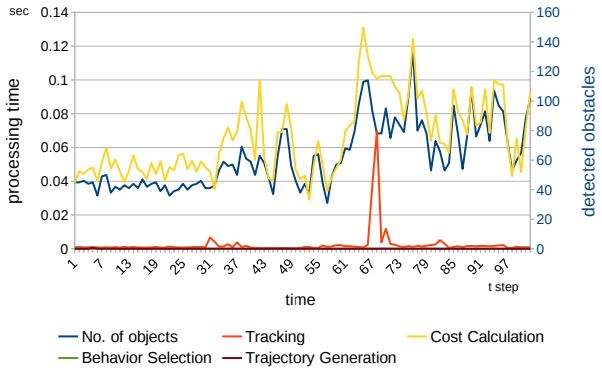


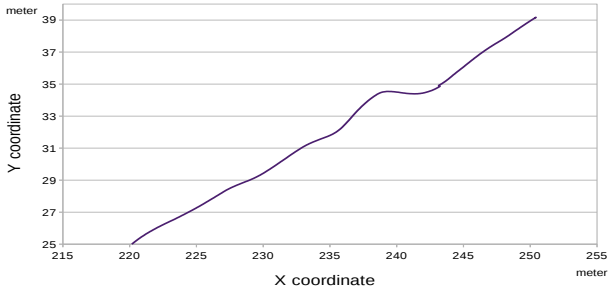
Fig. 26. Performance results of campus field test. *Y*-axis on the left shows processing time used for calculating (obstacle tracking, trajectory cost, behavior selection, and trajectory generation) in seconds, with relation to number of detected obstacles represented by right *Y*-axis.

tecting an average of 62 obstacles and an average total of 1,255 contour points (**Fig. 26**). For each iteration we calculate the obstacles contours, track the obstacles using a Kalman filter, calculate the costs and then generate new roll-outs if needed. In **Fig. 26** 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.

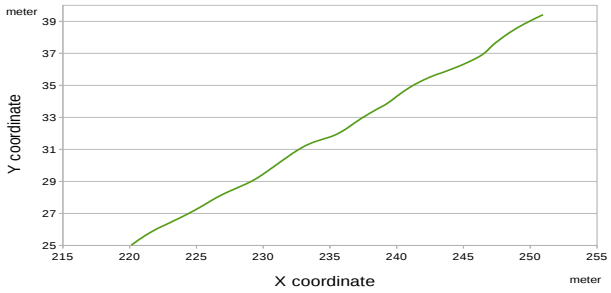
7.5. Accuracy

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 [e] on Youtube. We tested several parameters using the same route section illustrated in **Fig. 27**. 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 roll-in parameters, we were able to achieve a smoother

Fig. 27. A narrow pathway was used in the campus testing area to evaluate performance when avoiding nearby obstacles.



(a)



(b)

Fig. 28. 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.

driving path when traveling through this section. **Fig. 28** shows data from several trials while navigating through the section shown in **Fig. 27**.

8. Conclusion

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 paper 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 of this paper is to provide OpenPlanner 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.

Finally, we are open to suggestions and cooperative projects in order to continue to improve OpenPlanner, the latest version of which can be found under “feature/OpenPlanner-dev” at the Autoware GitHub website [a]. Additional illustrative videos and field experiments can be seen on the OpenPlanner channel [e].

Acknowledgements

This research is supported by the Center of Innovation Program (Nagoya-COI; Mobility Society leading to an Active and Joyful Life for Elderly) from Japan Science and Technology Agency, and JST Grants (Program on open innovation platform with enterprises, research institute and academia).

References:

[1] W. Burgard, A. Cremers, D. Fox, D. Hanel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "The interactive museum tour-guide robot," Proc. of the Fifteenth National Conf. on Artificial Intelligence (AAAI-98), 1998.

[2] Y. Morales, E. Takeuchi, A. Carballo, A. Aburadani, and T. Tsubouchi, "Autonomous robot navigation in outdoor cluttered pedestrian walkways," J. of Field Robotics, Vol.26, No.8, pp. 609-635, 2009.

[3] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard, "Autonomous robot navigation in highly populated pedestrian zones," J. of Field Robotics, 2014.

[4] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Stanley: The robot that won the DARPA grand challenge," J. of Field Robotics, Vol.23, No.1, pp. 661-692, June 2006.

[5] C. Urmson, J. Anhalt, H. Bae, J. D. Bagnell, C. Baker, R. E. Bittner, T. Brown, M. N. Clark, M. Darms, D. Demitrish, J. Dolan, D. Duggins, D. Ferguson, T. Galatali, C. M. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. Howard, S. Kolski, M. Likhachev, B. Litkouhi, A. Kelly, M. McNaughton, N. Miller, J. Nickolaou, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, V. Sadekar, B. Salesky, Y.-W. Seo, S. Singh, J. M. Snider, J. C. Struble, A. T. Stentz, M. Taylor, W. R. L. Whittaker, Z. Wolkowicki, W. Zhang, and J. Ziglar, "Autonomous driving in urban environments: Boss and the urban challenge," Special Issue on the 2007 DARPA Urban Challenge, Part I, J. of Field Robotics, Vol.25, No.1, pp. 425-466, June 2008.

[6] 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, Nov. 2015.

[7] 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," ICRA workshop on open source software, Vol.3, Issue 3.2, Dec. 2009.

[8] P. Weber and M. (M.) Haklay, "OpenStreetMap: User Generated Street Maps," IEEE Pervasive Computing, Vol.7, pp. 12-18, October-December 2008. doi: 10.1109/MPRV.2008.80.

[9] S. LaValle, "Motion planning: The essentials," IEEE Robotics Automation Magazine, Vol.18, No.1, pp. 79-89, 2011.

[10] S. LaValle, "Motion planning: Wild frontiers," IEEE Robotics Automation Magazine, Vol.18, No.2, pp. 108-118, 2011.

[11] E. W. Dijkstra, "A note on two problems in connexion with graphs," Numerische Mathematik, Vol.1, pp. 269-271, 1959.

[12] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," IEEE Trans. on Systems, Science, and Cybernetics, Vol.SSC-4, No.2, pp. 100-107, 1968.

[13] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime replanning algorithm," Proc. of the Int. Conf. on Automated Planning and Scheduling 2005 (ICAPS 2005), pp. 262-271, 2005.

[14] S. Koenig and M. Likhachev, "D*lite," Eighteenth National Conf. on Artificial Intelligence, Menlo Park, CA, USA: American Association for Artificial Intelligence, pp. 476-483, 2002.

[15] P. Beeson, N. K. Jong, and B. Kuipers, "Towards Autonomous Topological Place Detection Using the Extended Voronoi Graph," Proc. of the 2005 IEEE Int. Conf. on Robotics and Automation, pp. 4373-4379, April 2005.

[16] J. J. Park and B. Kuipers, "A smooth control law for graceful motion of differential wheeled mobile robots in 2D environment," 2011 IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 4896-4902, May 2011.

[17] P. Czerwionka, M. Wang, and F. Wiesel, "Optimized route network graph as map reference for autonomous cars operating on German autobahn," 2011 5th Int. Conf. on Automation Robotics and Applications (ICARA), pp. 78-83, December 2011.

[18] C. Chen, A. Gaschler, M. Rickert, and A Knoll, "Task Planning for Highly Automated Driving," 2015 IEEE Intelligent Vehicles Symposium (IV2015), 2015.

[19] R. Simmons, "The Curvature-Velocity Method for Local Obstacle Avoidance," Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA 1996), pp. 3375-3382, 1996.

[20] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," IEEE Robotics and Automation Magazine, Vol.4, No.1, pp. 23-33, 1997.

[21] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," IEEE Int. Conf. on Robotics and Automation, pp. 341-346, 1999.

[22] Y. Morales, A. Watanabe, F. Ferreri, J. Even, T. Ikeda, K. Shinozawa, T. Miyashita, and N. Hagita, "Including Human Factors for Planning Comfortable Paths," 2015 IEEE Int. Conf. on Robotics and Automation (ICRA 2015), pp. 6153-6159, May 2015.

[23] X. Li, Z. Sun, A. Kurt, and Q. Zhu, "A Sampling-Based Local Trajectory Planner for Autonomous Driving along a Reference Path," Proc. of 2014 IEEE Intelligent Vehicles Symposium (IV 2014), 2014.

[24] R. Ding and W. Chen, "Moving object tracking in support of unmanned vehicle operation," 2013 19th Int. Conf. on Automation and Computing (ICAC), pp. 1-6, 2013.

[25] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," IEEE Robotics and Automation Magazine, Vol.19, No.4, pp. 72-82, December 2012.

[26] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," Int. J. of Robotics Research, Vol.30, No.7, 2011.

[27] Y. Saad, "Iterative methods for sparse linear systems (2nd ed.)," Chapter 6, SIAM, 2003. ISBN 978-0-89871-534-7.

[28] R. C. Coulter, "Implementation of the pure-pursuit path tracking algorithm," Tech. Rep. CMU-RI-TR-92-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, Pa, USA, 1992.

[29] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," Proc. of the First Int. Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08), Chicago, IL, Menlo Park, CA, 2008.

[30] S. Muramatsu, T. Tomizawa, S. Kudoh, and T. Suehiro, "Development of Intelligent Mobile Cart in a Crowded Environment – Robust Localization Technique with Unknown Objects," J. of Robotics and Mechatronics, Vol.26, No.2, pp. 204-213, 2014.

[31] H. Bruyninckx, "Open robot control software: The OROCOS project," Proc. of 2001 IEEE Int. Conf. on Robotics and Automation (ICRA'01), Seoul, Korea, 2001.

Supporting Online Materials:

[a] https://github.com/CPFL/Autoware [Accessed July 15, 2017]

[b] http://www.tsukubachallenge.jp [Accessed July 15, 2017]

[c] D. V. Lu and M. Ferguson, "Navigation Stack," http://wiki.ros.org/navigation [Accessed July 15, 2017]

[d] https://github.com/open-rdc [Accessed July 15, 2017]

[e] https://www.youtube.com/playlist?list=PLVAImlqqGbr5G1EjscMom6_JcwjxCMX2g [Accessed July 15, 2017]

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 ROS nodes, way planner and DP planner. Both nodes use two shared libraries, utilityh.so and plannerh.so, which contain reusable functionality for all planning tasks. Both doxygen-based documentation and wiki pages are provided for these nodes/libraries. Since the OpenPlanner ROS nodes basically use these two libraries as the system's core functionality, the basic planner could be used outside ROS with any other platform, for example as in [31, c].

A.1. Way Planner Node

The way 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 way planner node include a vector map, current position and goal

Table 4. 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.

position. Output is the shortest path or paths from the map after taking into account predefined traffic costs, as shown in **Figs. 8** and **9**.

The launch file for the way planner node has several parameters that enable users to control the behavior and performance of the node, as shown in **Table 4**. Additionally, the way planner global planning node could be used with any local planner. It is the responsibility of the task planner to invoke the way 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.2. DP Planner Node

The DP planner node 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 DP planner node is provided in **Table 5**.

Appendix B. Algorithms

In this section we list the main algorithms used in OpenPlanner. The algorithm in **Table 6** fix errors in the vector map raw data by interpolating points along the path with fixed distance. In **Tables 7** and **8** the main algorithm for finding global path between start and goal points. The algorithm in **Tables 9** and **10** used to generate roll out trajectories for the local planner.

Table 5. 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

Table 6. Algorithm for interpolating path points with fixed density.

Algorithm 1 Adjust path waypoints density (maxDistance)

```
remainingDistance = 0
for I = 0 to pathSize() - 1
    d := distance(Pi, Pi+1)
    nNewPoints := d / maxDistance
    if remainingDistance == 0
        newPath := Pi
    end if
    a = angle(Pi+1, Pi)
    for j = 0 to nNewPoints -1
        Pix' := Pix' + maxDistance * cos(a)
        Piy' := Piy' + maxDistance * sin(a)
        newPath := Pi
    end for
end for
return newPath
```

Table 7. 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 8. Detailed algorithms of procedure in **Table 7**.

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 9. Algorithm to generate roll out trajectories used in local planner.

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

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

Table 10. Detailed algorithms of procedure in **Table 9**.

Algorithm 3 procedures

```
01: procedure ExtractPlanningSection(sPose, maxDistance, globalPath)
02:     index := FindClosestWaypoint(sPose, globalPath)
03:     while distance LESSTHAN maxDistance
04:         pathSection := globalPath(index++)
05:     end while
06:     FixPathDensity(pathSection)
07:     SmoothPath(pathSection)
08:     return pathSection
09: end procedure
10:
11: procedure SampleTrajectories(sPose, pathSection, nRollOuts, carTipMargin,
rollInMargin, rollOutMargin)
12:     rollOuts := CreateList(nRollOuts)
13:     for i = 0 to nRollOuts
14:         rollOuts[i] = AddCarTipSection(pathSection, carTipMargin)
15:         distanceFromCenter = rollOutDensity * (i - nRollOuts/2)
16:         rollOuts[i] = SampleRollInSection(pathSection, distanceFromCenter,
rollInMargin)
17:         rollOuts[i] = AddRollOutSection(pathSection, rollOutMargin)
18:         SmoothPath(rollOuts[i])
19:     end for
20:     return rawRollOuts
21: end procedure
22:
23: procedure SmoothRollOuts(rawRollOuts)
24:     return rollOuts
25: end procedure
```

Name:
Hatem Darweesh

Affiliation:
Department of Intelligent Systems, Graduate School of Informatics, Nagoya University

Address:
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan

Brief Biographical History:
1998-2002 B.Sc., Faculty of Computers and Information Sciences, Ain Shams University
2002-2013 Teaching Assistant, Moder Academy in Maadi
2004-2009 M.Sc., Faculty of Computers and Information Sciences, Ain Shams University
2004-2010 Co-Founder, NRG Solutions, Egypt
2013-2016 Robotics Engineer, ZMP Inc., Japan
2016- Ph.D. Student, Graduate School of Information Science, Nagoya University

Name:
Eijiro Takeuchi

Affiliation:
Department of Intelligent Systems, Graduate School of Informatics, Nagoya University

Address:
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan

Brief Biographical History:
2008-2015 Assistant Professor, Tohoku University
2015-2016 Designated Associate Professor, Nagoya University
2016- Associate Professor, Nagoya University

Main Works:
• "A 3-D Scan Matching using Improved 3-D Normal Distributions Transform for Mobile Robotic Mapping," 2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS2006), Beijing, China, pp. 3068-3073, 2006.
• "Development of an Intelligent Senior-Car in a Pedestrian Walkway," Advanced Robotics, Vol.26, No.14, pp. 1577-1602, Jul. 2012.

Membership in Academic Societies:
• The Japan Society of Mechanical Engineering (JSME)
• The Robotics Society of Japan (RSJ)
• The Society of Instrument and Control Engineers (SICE)
• The Institute of Electrical and Electronics Engineers (IEEE)

Name:
Kazuya Takeda

Affiliation:
Department of Media Science, Graduate School of Information Science, Nagoya University

Address:
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan

Brief Biographical History:
1983/1985 Received B.E./M.E. degrees in Electrical Engineering from Nagoya University
1986-1989 Advanced Telecommunication Research laboratories (ATR)
1987-1988 Visiting Scientist, MIT
1989-1995 Researcher and Research Supervisor, KDD Research and Development Laboratories
1994 Received Doctor of Engineering degree from Nagoya University
1995-2003 Associate Professor, Faculty of Engineering, Nagoya University
2003- Professor, Department of Media Science, Graduate School of Information Science, Nagoya University

Main Works:
• media signal processing and its applications, which include spatial audio, robust speech recognition and driving behavior modeling

Name:
Yoshiki Ninomiya

Affiliation:
Intelligent Vehicle Research Division, Institute of Innovation for Future Society, Nagoya University

Address:
609 National Innovation Complex (NIC), Furo-cho, Chikusa-ku, Nagoya 464-8601, Japan

Brief Biographical History:
1981 Received B.S. from Nagoya University
1983 Received M.S from Nagoya University
1983-2003 Researcher, Toyota Central Lab.
2008 Received Ph.D. from Nagoya University
2014- Designated Professor, Nagoya University

Name:
Adi Sujiwo

Affiliation:
Department of Information Engineering, Graduate School of Informatics, Nagoya University

Address:
609 National Innovation Complex (NIC), Furo-cho, Chikusa-ku, Nagoya 464-8601, Japan

Brief Biographical History:
2009-2011 Magister of Computer Science, University of Indonesia
2011-2014 System Engineer, Bogor Agricultural University, Indonesia
2014- Ph.D. Student, Nagoya University

Name:
Luis Yoichi Morales

Affiliation:
Driving Scene Understanding Research Division, Institute of Innovation for Future Society, Nagoya University

Address:
609 National Innovation Complex (NIC), Furo-cho, Chikusa-ku, Nagoya 464-8601, Japan

Brief Biographical History:
2006 Received M.Eng., Intelligent Robot Laboratory, Tsukuba University
2009 Received Ph.D., Intelligent Robot Laboratory, Tsukuba University
2010-2016 Researcher, ATR Intelligent Robotics and Communication Laboratories
2016- Designated Associate Professor, Nagoya University

Membership in Academic Societies:
• The Institute of Electrical and Electronics Engineers (IEEE) Robotics and Automation Society
• The Robotics Society of Japan (RSJ)

Name:
Naoki Akai

Affiliation:
Driving Scene Understanding Research Division, Institute of Innovation for Future Society, Nagoya University

Address:
609 National Innovation Complex (NIC), Furo-cho, Chikusa-ku, Nagoya 464-8601, Japan

Brief Biographical History:
2013-2016 Doctor Student, Utsunomiya University
2016- Designated Assistant Professor, Nagoya University

Main Works:
• "Development of mobile robot "SARA" that completed mission in real world robot challenge 2014," J. of Robotics and Mechatronics, Vol.27, No.4, pp. 327-336, 2015.
• "Gaussian processes for magnetic map-based localization in large-scale indoor environments," IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 4459-4464, 2015.
• "Robust localization using 3D NDT scan matching with experimentally determined uncertainty and road marker matching," IEEE Intelligent Vehicles Symposium, pp. 1364-1371, 2017.

Membership in Academic Societies:
• The Institute of Electrical and Electronics Engineers (IEEE)
• The Japan Society of Mechanical Engineering (JSME)
• The Robotics Society of Japan (RSJ)
• Society of Instrumentation and Control Engineering (SICE)

Name:
Tetsuo Tomizawa

Affiliation:
Department of Computer Science, National Defense Academy of Japan

Address:
1-10-20 Hashirimizu, Yokosuka-City, Kanagawa 239-8686, Japan

Brief Biographical History:
2005- Research Fellow, Japan Society for the Promotion of Science (JSPS)
2007- Post-Doctoral Research Scientist, National Institute of Advanced Industrial Science and Technology (AIST)
2009- Assistant Professor, The University of Electro-Communications
2016- Lecturer, National Defense Academy of Japan
2017- Visiting Associate Professor, Nagoya University

Main Works:
• "A Person Detection Method Using 3D Laser Scanner – Efficient Grouping Method of Point Cloud Data –," J. of Robotics and Mechatronics, Vol.27, No.4, pp. 374-381, Aug. 2015.
• "Development of an Intelligent Senior-Car in a Pedestrian Walkway," Advanced Robotics, Vol.26, No.14, pp. 1577-1602, Jul. 2012.

Membership in Academic Societies:
• The Japan Society of Mechanical Engineers (JSME)
• The Institute of Electrical and Electronics Engineers (IEEE) Robotics and Automation Society
• The Robotics Society of Japan (RSJ)

Name:
Shinpei Kato

Affiliation:
Graduate School of Information Science and Technology, The University of Tokyo

Address:
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8654, Japan

Brief Biographical History:
2004 Received B.S. degree from Keio University
2006 Received M.S. degree from Keio University
2008 Received Ph.D. degree from Keio University
2009-2012 Worked at The University of Tokyo, Carnegie Mellon University, and University of California, Santa Cruz
2012-2016 Associate Professor, Nagoya University
2016- Associate Professor, Graduate School of Information Science and Engineering, The University of Tokyo