Path Planning Combining Improved Rapidly-Exploring Random Trees with Dynamic Window Approach in ROS

Jianxun Wang[1,2]
Hubei Collaborative Innovation Center for Advanced Steels
School of Information Science and Engineering
[1]Wuhan University of Science and Technology, 430081, Wuhan, China
wodexiaoxianfeng@gmail.com

Shiqian Wu[1]
Hubei Collaborative Innovation Center for Advanced Steels
Key Laboratory of Metallurgical Equipment and Control Technology, Ministry of Education
[1]Wuhan University of Science and Technology, 430081, Wuhan, China
shiqian.wu@wust.edu.cn

Huiyun Li*[2]
Shenzhen Engineering Laboratory on Autonomous Driving
[2]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
1068 Xueyuan Avenue
Shenzhen, P.R.China
hy.li@siat.ac.cn

Jie Zou[2]
Shenzhen Engineering Laboratory on Autonomous Driving
[2]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
1068 Xueyuan Avenue,
Shenzhen, P.R.China
jie.zou@siat.ac.cn

***Abstract*—Path planning is a fundamental research area in robotics. Sampling-based methods have been extended further away from basic robot planning into further difficult scenarios and diverse applications for their efficient solution. Issues occurred where they only offer a path rather than velocity commands; the changing situation of environment is hard to tackle. This paper presents a novel integrated approach of creating path for robot navigation with dynamic constrains. In the proposed algorithm, a biased Rapidly-Exploring Random Trees (RRT) is utilized to find a global path in the configuration space. Then, the Dynamic Window Approach (DWA) is performed over the path to calculate translational and rotational velocity commands for the robot. Performance of the proposed method is tested and validated using Robot Operating System (ROS). Simulations show that the proposed methodology achieves efficient and smooth path for the robot under the dynamic constrains.**

Keywords—Path Planning; RRT; DWA; ROS; Robot Navigation

I. INTRODUCTION

Generally, an intelligent robot should have the ability to autonomously construct an accessible trajectory between two positions through complex environment. Path planning can be defined as: provided a mobile base and a model of its environment, generating a dynamic feasible path with cluttered obstacles that connects a valid given start and goal configuration with optimal indexes such as minimum distance, away from obstacles, minimum run time, maximum speed and minimum energy consumed [1]. Autonomous mobile robot path planning or navigation is one of the most important applications for robot control systems and the field has attracted remarkable attention from number of researchers [2]-[4].

Path planning can be partitioned into global and local method. Previous methods use priori information of obstacles and generate trajectories between the initial position and the goal position inside collision free space. Probabilistic road maps (PRMs) [5], RRT [6] are the most popular in this category. In [2] and [7] descriptions, applications and improvements of these algorithms are reviewed. They assume a static world and do not take sensory information into consideration. The later planning, also called reactive path planning, methods is divided into two categories, the first kind computes appropriate robot heading angle for obstacles free navigation. Artificial Potential Field (APF) [8], VFH+ [9], Obstacle Restriction Method (ORM) [10] and Follow the Gap Method (FGM) [11] are popular methods in this category. The detail review was given in [12]. Although these approaches efficiently generate direction outputs, they are not sufficient for taking the robot dynamics into account. The other uses dynamic properties of robots in order to perform obstacle free navigation. As described in [12], the Curvature Velocity Method (CVM) [13], Velocity Obstacles method (VO) [14] and Dynamic Window Approach (DWA) [15], [16] belong to this category. They presume the robot travels as arcs.

The authors of [17]-[19] proposed planning methods combining global and local planner to get better results. The method of [17] combines Distance Transform Path Planner and Potential Field navigation method. The advantage of the above two methods has been developed furthermore several of their weakness has been eliminated. Global and local path planning problem was combined into a single search using a combined 2-D and higher dimensional state-space in [18]. A methodology for the path planning calculation was presented in [19] which splits path planning problem into two stages. Stage one gives a coarse global path and stage two enhances the precision by detecting system mainly based on computer vision. However, velocity commands cannot be obtained directly because of only trajectories provided. Astar [20] and Dijkstra [21] algorithm have been packaged into ROS Navigation Stack. Although they offer optimal path by searching grid map, efficiency cannot be ensured when encountering large-scale environment.

In this paper, biased RRT is utilized as a global path planner which randomly samples the configuration space to generate a collision free path with bias to the goal region whilst DWA is employed to sample velocity space to provide a local trajectory with admissible velocity pair for mobile robot, which takes robot dynamic into consideration. Biased RRT allows a fast planning in semi structured spaces by executing a random search through navigation area. The output of DWA can prompt the robot to move diametrically. Simulation is conducted with Navigation Stack of ROS, in which the biased RRT algorithm is integrated as a global planner plugin and DWA a local planner.

The rest of this paper are arranged as follows. Section □ overviews RRT algorithm and DWA, the proposed method RRT-DWA is introduced in section □. Simulation results and tests are detailed in section □. Conclusion of this paper is described in section □.

978-1-5386-3758-6/18/$31.00 ©2018 IEEE

□. RELATED WORK

A. Rapidly-Exploring Random Trees

Path planning will generally be considered as a search in a configuration space, $\mathcal{C}$, in which each $q \in \mathcal{C}$ specifies the position of one or more agents(mobile robots, vehicles, arms and so on) in a 2D or 3D world. The state of an agent is defined by a metric ρ, such as distance, time, energy on $\mathcal{C}$. It's assumed that the set of configurations which those agents do not intersect with any static obstacles is free space, denote as $\mathcal{C}_{free}$, otherwise denote as $\mathcal{C}_{obs}$. In order to determine whether a q is belong to $\mathcal{C}_{free}$ or $\mathcal{C}_{obs}$, collision detection algorithm is used. The mission of a single-query path planning is to find a continuous path from a start point or region ($q_{init} \in \mathcal{C}_{free}$) to a goal point or region ($q_{goal} \in \mathcal{C}_{free}$).

Rapidly-Exploring Random Tree was introduced in [6] as a randomized data structure and specifically designed to handle nonholonomic constrains and high degrees of freedom with quick search speed. The key advantage of RRT is the bias of its expansions toward unexplored portions of the configuration space. A RRT will be constructed when all of its vertices are $q \in \mathcal{C}_{free}$ and each edge of it will correspond to a path that lies entirely in $\mathcal{C}_{free}$.

For a given start and goal configuration q_{init}, q (a ε region around q_{goal}) $\in \mathcal{C}_{free}$, a RRT, $\mathcal{T}$, with K vertices is constructed as shown in Table 1:

Table 1: The Basic RRT algorithm

```
GENERATE_RRT(q_init, K, Δt)
1   T.init(q_init);
2   for k=1 to K do
3       q_rand ← RANDOM_POSITION();
4       q_near ← NEAREST_NEIGHBOR(q_rand, T);
5       u ← SELECT_INPUT(q_rand, q_near);
6       q_new ← NEW_POSITION(q_near, u, Δt);
7       T.add_vertex(q_new);
8       T.add_edge(q_near, q_new, u);
9       if q_new=q
10          Return Reached
11      else
12          Return Advanced
13      Return Trapped
14  Return T
```

The first vertex of $\mathcal{T}$ is $q_{init} \in \mathcal{C}_{free}$. In every single iteration, a random configuration, q_{rand}, is selected from $\mathcal{C}$ (it is assumed that $\mathcal{C}$ is bound). Step 4 finds the closest vertex in $\mathcal{T}$ to q_{rand} in terms of ρ (a distance metric). Step 5 selects an input, u, that minimizes the distance between q_{near} and q_{rand}, and ensures that configuration remains in $\mathcal{C}_{free}$. Collision detection can be performed quickly using incremental distance computation algorithms. By applying u in q_{near}, q_{rand}, a potential new position is chosen. There three situations can occur: Reached, in which q_{goal} has been added to $\mathcal{T}$, a path found; Advanced, in which a new vertex $q_{new} \notin q_{goal}$ is added to the RRT; Trapped, in which the chosen new vertex is rejected due to it does not belong to $\mathcal{C}_{free}$.

B. Dynamic Window Approach

Dynamic Window Approach (DWA) is a velocity search method which the search for commands controlling translation and rotational velocity of robot is carried out directly in space of velocity to reactive collision avoidance for mobile robots. And besides that it also correctly and in an elegant way incorporates dynamics of the robot. The method is divided into two parts.

Search space

The search space of possible velocities is divided into three steps. First, choosing two-dimensional velocity search space in which only circular trajectories determined uniquely by pairs *(v, w)* of translational and rotational velocities. Second, finding admissible pairs that ensure-only safe trajectories-the robot stop before it reaches the closest obstacle on the corresponding curvature. Let $\dot{V}_b$ and $\dot{w}_b$ be the accelerations for breakage. The set of admissible V_a is defined as Eq. (1)

$$V_a = \left\{(v,w) \mid v \le \sqrt{2 \cdot dist(v,w) \cdot \dot{V}_b} \wedge w \le \sqrt{2 \cdot dist(v,w) \cdot \dot{w}_b}\right\} \quad (1)$$

Third, determining dynamic window by restricting admissible velocities (V_a) to those that can be reached within a short time interval given limited accelerations of robot. Then V_d with time interval t is defined as Eq. (2)

$$V_d = \{(v,w) | v \in [V_a - \dot{v} \cdot t, V_a + \dot{v} \cdot t] \wedge w \in [w_a - \dot{w} \cdot t, w_a + \dot{w} \cdot t]\} \quad (2)$$

Let *Vs* denote the space of possible velocities (constrains generated by the robot, the maximum and minimum velocity), then resulting search space area *Vr* is defined as

$$V_r = V_s \cap V_a \cap V_d$$

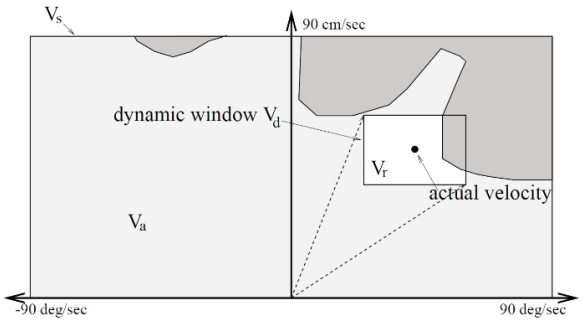


Fig. 1. Resulting velocity space [15]

Optimization: objective function

In part A, we determine the resulting search space V_r. Obviously, there are lots of velocity pairs among area V_r. An objective function Eq. (3) is proposal to criteria each pair.

$$G(v,w) = \sigma(\alpha \cdot heading(v,w) + \beta \cdot dist(v,w) + \gamma \cdot velocity(v,w)) \quad (3)$$

- Target heading

  The function *heading(v, w)* evaluates difference between angle of target point and angle of predicted position which

robot moves from current position using current velocity after some time interval t. As depicted in Fig. 2, θ is the metric of the function, and it is given by *180-*θ.

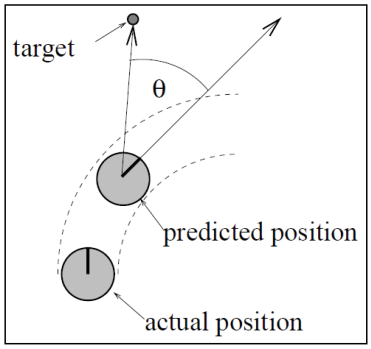


Fig. 2. Angle θ to the target [15]

- Clearance
  The function *dist(v, w)* implies the distance to closest obstacle which intersects with current trajectory of robot. If no intersects, the value of this function is set to a large constant.
- Velocity
  The function *velocity(v, w)* calculates linear velocity values in the velocity set.
- Smoothing
  The operation used is to normalize all of the three components of the objective function to [0, 1], which calculates the coefficients α,β,γ. This can help deal with the situation that some of the components occur discontinuous parts which makes some items take leading role.

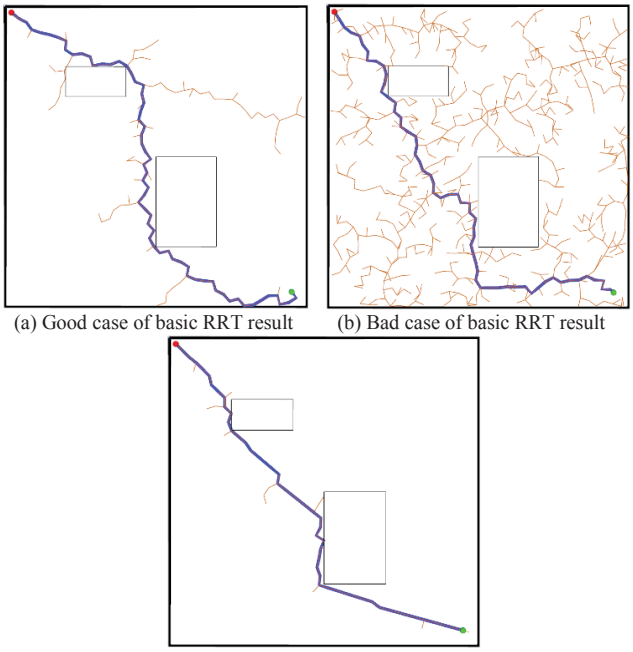
Maximizing the objective function results in safe trajectories to reach target as fast as possible.

III THE PROPOSED METHOD

A. *Selection of bias of goal region in RRT*

As mentioned before, RRT is a sample based single-query path planning algorithm which can quickly search high-dimensional spaces that have both algebraic constrains and differential constrains. The key idea is the bias of exploration toward unexplored potation of the configurations. This advantage makes uniform distribution for sampling, meaning all of the configuration space has same probabilistic to be selected. On the contrary, duo to the uniform probabilistic, quite a partition of expanding of RRT is necessary and a waste of time.

We find that performance of the search will be enhanced if to bias the growth of RRT towards goal instead of exploring the whole space. This can be achieved by selecting the point on direction of goal with some probability *p*. Then the biased RRT chooses goal location with probability *p* and random points are chosen with probability *1-p*. The value of p is a design parameter whose changing depends on the environment. If densely cluttered with obstacle, it's a good idea to decrease the value otherwise higher value will reduce the time for planning a collision free path.



(a) Good case of basic RRT result (b) Bad case of basic RRT result

(c) Biased RRT result

Fig. 3. Basic RRT results (upper), Biased RRT result (lower). Start and goal are marked as red and green point. Obstacle area is two black rectangles. The extension is brown line and the final path is blue line. The biased RRT shows flexible path using less time.

Fig. 3 depicts basic and biased RRT path planning results. In Fig. 3 (a), the basic RRT find a collision free path without exploring the whole space but more cases can occur as shown in Fig. 3 (b). This phenomenon is caused by the fact that the basic RRT always selects points randomly for extending almost the whole configuration space. In Fig. 3 (c), the biased RRT is capable of digging out a more flexible and shorter path with less time because of traveling less unnecessary exploring area compared with the basic RRT.

B. *Velocity commands provision by DWA*

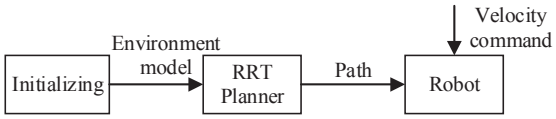


Fig. 4. Path planner without DWA

As explained in part A, the biased RRT (or the basic RRT) can generate a flexible collision free path. The system configuration is simplified as Fig. 4. The initializing block provides information required by RRT planner, such as position of obstacles, footprint of robot, start and goal of this planning and bias of goal region. After initializing, RRT planner will offer a path without intersections with obstacles and the path is sent to the robot. Velocity command is given by operators so that robot could follow the path smoothly. This introduces issues how to ensure the value of velocity, how to ensure robot follow the path preciously, how to ensure interface with sensory

message of robot if provided and etc. The key point is that RRT planner cannot supply a collision free path at the same time furnish velocity commands determined by obstacles and path.

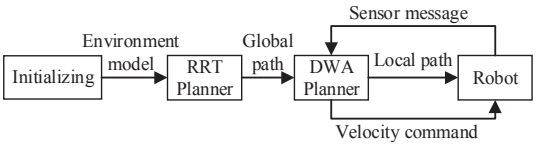


Fig. 5. The proposed path planner

As shown in Fig. 5, DWA planner is added behind RRT planner. The initializing is identical with the above besides dynamic parameters of robot like the maximum velocity, the minimum rotation radius subjoined. The RRT planner will search a flexible path from start to goal and then the path is sent to DWA planner. Simultaneously sensor messages of robot is transmitted, DWA planner will calculate best translational and rotational velocity to robot.

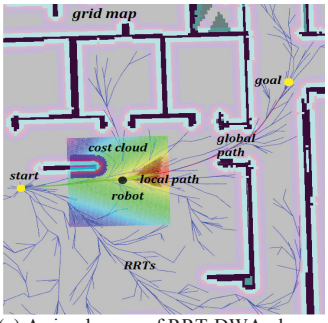
Table 2. Algorithm of calculate velocity

CACULATE_VELOCITY(*v,w*)	
1	Initializing(*map, V_max, R_min*);
2	global_path ← RRT_Planner();
3	sensor_messages ← Robot;
4	if the trajectory without obstacles
5	Obs←cost_sum(*cost_cloud*);
6	Gdist←distance(*end_point, goal*);
7	Pdist←distance(*end_point, global_path*);
8	*v*←translational part of generates the trajectory;
9	minimum *C(t)*;
10	Return (*v,w*)

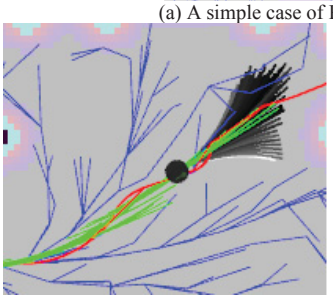
The DWA is directly sampling velocity space and takes dynamic constrains into consideration. The process of calculation of velocity is illustrated in Table 2. Step initializing planner with a map and robot dynamic parameters. Step 2 to Step 3 receives a global path provided by RRT planner and sensor messages from robot online respectively. For each sampled velocity, perform forward simulation from robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time and trajectories is left. The sampled trajectories that have intersections with obstacles will be eliminated directly.The objective function replaced by a cost function [16]:

$$C(t) = \alpha \cdot Obs + \beta \cdot Gdist + \gamma \cdot Pdist + \sigma \cdot \frac{1}{v^2} \quad (4)$$

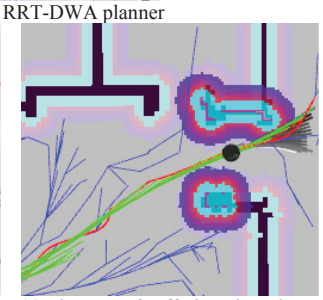
where *Obs* is the sum of costs through which the trajectory passes (as shown in Fig. 6 (a), cost cloud); *Gdist* and *Pdist* are estimated shortest distances from the endpoint of trajectory to goal and global path, respectively; and *v* is the translational component of velocity command that produces trajectory. Step 5 to Step 8 determines parameters required by the cost function which is minimized in Step 9. The chosen trajectory has the minimum value of Eq. (4) resulting in some properties: (□) a safe route far from obstacles; (□) biasing to the goal; (□) staying close with the optimal path; (□) having great speed. Trajectories that collide with obstacles will be discarded straitly.



(a) A simple case of RRT-DWA planner



(b) Selecting of sampled trajectories



(c) The case of suffering obstacles

Fig. 6. RRT-DWA path planner

As shown in Fig. 6 (a), after specifying a start and goal (yellow point) configuration, RRT algorithm searches the space (blue line) until a path found (red line), then DWA computes a flexible path (green line) to follow the red line avoiding obstacles online. Task ends by the robot reaching goal configuration. In Fig. 6 (b), black arcs indicate the sampled trajectories, and green arcs are selected to generate velocity commands. When encountered with obstacles, DWA will choose the best velocities as shown in Fig. 6 (c).

□. SIMULATION RESULTS

According all of the analysis above, RRT can provide an efficient path in large scale and DWA can straightway give proper velocity commands to robot. So, we assign RRT a global path planner which generates a referential global path for supplying some more specific information. And then DWA computes translational velocity and rotational velocity for the agent. Finally we implement this pipe line on Robot Operating System (ROS), for navigation stack taking the key role. Technical details will be introduced in rest of this section. There are brief instructions about ROS and its Navigation Stack as following.

A. Robot Opera System

ROS is a BSD-licensed system for controlling robotic components from a PC. It provides a structured communications layer above the host operating systems of a heterogeneous compute cluster. A ROS system is comprised of a number of independent nodes, each of which communicates with the other nodes using a *publish/subscribe* messaging model.

Note that nodes in ROS do not have to be on the same system or even of the same architecture. This makes ROS really flexible and adaptable to the needs of users.

B. ROS Navigation Stack

In order to achieve navigation task, the Navigation Stack is used to integrate mapping, localization, and path planning together. It takes in information from *odometry*, sensor streams, and goal position to produce safe velocity commands and send it to the mobile base. The path planning block is achieved by *global_planner* and *local_planner*, in which map and sensor sources deliver to yield signals to drive robot.

C. Simulation details

The path planning part is performed in the *move_base* package from ROS Navigation Stack, divided into global and local planning modules which is a common strategy to deal with complex planning problem. The global path planner searches for a best path to the goal and local path planner, incorporating current sensor readings, issues actual commands to follow the global path while avoiding obstacles. The *move_base* package also maintains two costmaps, *global_costmap* and *local_costmap* to be used with the global and local planners respectively. Costmap used to store and maintain information in the form of occupancy grid about the obstacles in the environment and where the robot should navigate to. Costmap initialized with prior static map if available, then it will be updated using sensor data to maintain the information about obstacles in the world.

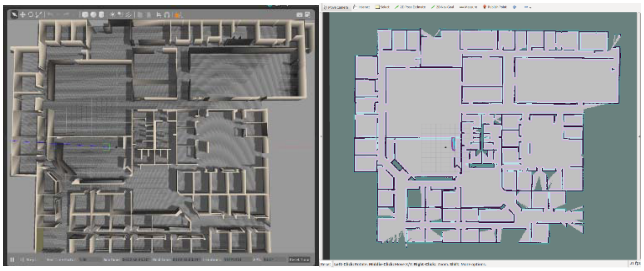


Fig. 7. Gazebo and RVIZ

First, we write a global path planner plugin to replace the default planner in Navigation Stack [20]. Second, simulation environment offered by *Gazebo* (Fig. 7, left) including a 3D simulator world of willow garage office and a mobile robot. Third, a grid map containing world information is loaded (the map is not essential part to Navigation Stack, but string with a prior map will have significant benefits on the performance) which is made by a ROS wrapper for *OpenSlam's Gmapping*. Furth, *RVIZ* (Fig. 7, right) is used to display the messages including map, path and so on. Fifth, we set robot initial position each iteration, the goal can be chosen as specify coordinates relative to start position (we select seven points as shown in Fig. 8, the yellow points). Finally, path planning algorithm start processing and there are some crucial parameters: maximum translation and rotation velocities is set as (0.5, 0.4); sim_time that DWA will use to predict is set as 2.0s; tolerance of goal is set as (0.1, 0.1), the previous representing position error and the later instructing theta.

Three performance metrics are considered to evaluate the planners: (1) Planning and Walking time, the planning time representing time resumed for finding a global path; the walking time representing time resumed for robot reach goal from initial position, (2) Path length, the global path found by global planner, (3) Average speed, the average speed of mobile base to reach goal from initial position. We run the planner 20 times for each tour and results are shown.

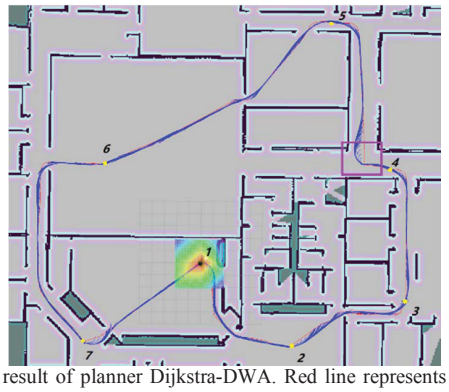


Fig. 8. The result of planner Dijkstra-DWA. Red line represents global path provided by Dijkstra and the blue one represents local path provided by DWA. Dijkstra can provide optimal path, but as marked in purple rectangle, a right-angle turn occurred. In addition, the planning is time consuming.

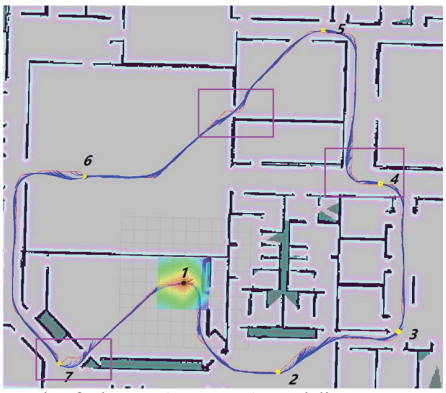


Fig. 9. The result of planner Astar-DWA. Red line represents global path provided by Astar and the blue one represents local path provided by DWA. Astar performs a higher planning speed. But the problem is as same as Dijkstra which is shown in purple rectangles.

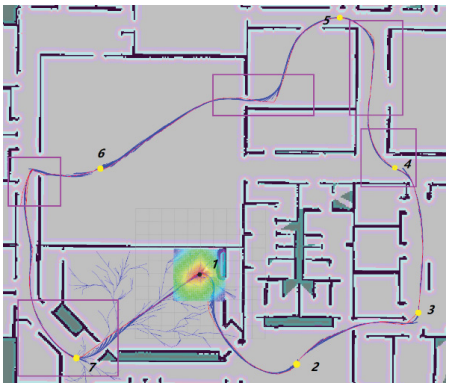


Fig. 10. The result of planner RRTs-DWA. Red line represents global path provided by biased RRT and the blue one represents local path provided by DWA. The time using in global planning is minimum but longest path.

Table 3: Planning time(ms) and Path length(m) over 20 simulations

Planner	Planning time	Path length
Dijkstra-DWA	91.4875	107.2616
Astar-DWA	82.3542	110.1180
RRT-DWA	60.2487	116.3364

Table 4: Walking time(s) and Average velocity (m/s) over 20 simulations

Planner	Walking time	Average velocity
Dijkstra-DWA	278.6739	0.3849
Astar-DWA	277.7251	0.3965
RRT-DWA	271.4751	0.4285

Dijkstra-DWA and Astar-DWA methods are tested as comparison to our RRT-DWA. In Fig. 8 and Fig. 9, an optimal global path is generated and the path is smoothly. But for finding shortest path, right-angle turn could be discorded and the time consumed is more as shown in Table 3. In Fig. 10, although the path found by biased RRT is not an optimal one, smoother trajectory and smaller turn angle make larger average velocity as shown in Table 4. The purple rectangle contents of Fig. 10 show better dynamic performance. Even though we have no plump evidence to prove the average distance away from obstacles of the proposed method, we believe the method has the property.

. CONCLUSION

In this paper, we combine global with local path planning method to efficiently generate a collision free path and at the same time velocity commands are sent to robot by associating sensor messages. The proposed method utilizes and improves two exist techniques, RRT and DWA, to get a smoother path with less time (27% quicker than Astar) and higher speed (8% faster than Astar) for robot. A bias of goal region is applied in RRT to speed up the algorithm and under the premise of guaranteeing the performance, the origin DWA is simplify with better properties.

We will focus on the follow issues in feature works. (1) To develop an optimal path finding algorithm has better performance, such as RRT*, D* Lite which are difficult to integrate into ROS planner plugin. (2) The agent is a circular model, more complicated model should be considered. (3) The environment we load is a static 2D grid map which in the real world is rarely exiting. And there are not any dynamic obstacle in our simulations. So, the path planning for robots problem still has a lot work to handle.

ACKNOWLEDGMENT

This work was partly supported by The National Natural Science Foundation of China (61267002, 41271362, 61371190, 61371190); Shenzhen Engineering Laboratory on Autonomous Driving, and Shenzhen S&T Funding JSGG20141020103523742.

REFERENCES

[1] Paden B, Čáp M, Yong S Z, Yershv D and Frazzoli E, "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles," IEEE Transactions on Intelligent Vehicles, 2016, 1(1):33-55.

[2] Karaman S, Frazzoli E, "Sampling-based optimal motion planning for non-holonomic dynamical systems," IEEE International Conference on Robotics and Automation. IEEE, 2013:5041-5047.

[3] M Jiang, Y Chen, W Zheng, H Wu, "Mobile robot path planning based on dynamic movement primitives," IEEE International Conference on Information and Automation. IEEE, 2017:980-985.

[4] Baizid K, Chellali R, Luza R, Vitezslav B and Arrichiello F. "RRS: Rapidly-Exploring Random Snakes a New Method for Mobile Robot Path Planning," 2016.

[5] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Trans. Robot. Autom.*, vol. 12, pp. 566–580, 1996.

[6] S. M. La Valle, "Rapidly-exploring random trees a new tool for path planning," Iowa State University, Technical Report No. 98-11, Oct. 1998.

[7] A.Vahidi and A. Eskandarian, "Research advances in intelligent collision avoidance and adaptive cruise control," Trans. Intell. Transp. Syst., vol. 4, pp. 143–153, 2003.

[8] Khatib, Oussama. "Real-time obstacle avoidance for manipulators and mobile robots," The international journal of robotics research 5.1 (1986): 90-98.

[9] Ulrich, I., Borenstein, J., "VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots," in Proceedings of the International Conference on Robotics and Automation (ICRA 98), Leuven, Belgium, May 1998.

[10] Minguez, Javier. "The obstacle-restriction method for robot obstacle avoidance in difficult environments," 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2005.

[11] Sezer, Volkan, and Metin Gokasan. "A novel obstacle avoidance algorithm: Follow the Gap Method," Robotics and Autonomous Systems 60.9 (2012): 1123-1134.

[12] Özdemir, Aykut, and V. Sezer. "A Hybrid Obstacle Avoidance Method: Follow the Gap with Dynamic Window Approach," *IEEE International Conference on Robotic Computing* IEEE, 2017:257-262.

[13] Simmons R., "The Curvature Velocity Method for Local Obstacle Avoidance," in Proceedings of the IEEE International Conference on Robotics and Automation, Minneapolis, April 1996.

[14] Gal, Oren, Zvi Shiller, and Elon Rimon. "Efficient and safe on-line motion planning in dynamic environments," Robotics and Automation, 2009. ICRA'09. IEEE International Conference on. IEEE, 2009.

[15] Fox D., Burgard W. and Thrun S., "The Dynamic Window Approach to Collision Avoidance," IEEE Robotics and Automation Magazine, 4:23-33, 1997.

[16] Gerkey, Brian P, and K. Konolige, "Planning and control in unstructured terrain," 2008.

[17] Wang, Lim Chee, L. S. Yong, and M. H. Ang. "Hybrid of global path planning and local navigation implemented on a mobile robot in indoor environment," *IEEE International Symposium on Intelligent Control* IEEE, 2002:821-826.

[18] Zhang, Haojie, J. Butzke, and M. Likhachev. "Combining global and local planning with guarantees on completeness," *IEEE International Conference on Robotics and Automation* IEEE, 2012:4500-4506.

[19] A. Signifredi, B. Luca, A. Coati, JS. Medina and D. Molinari, "A General Purpose Approach for Global and Local Path Planning Combination," *IEEE, International Conference on Intelligent Transportation Systems* IEEE, 2015.

[20] Hart, Peter E, N. J. Nilsson, and B. Raphael. "Correction to A Formal Basis for the Heuristic Determination of Minimum Cost Paths." *IEEE Transactions on Systems Science & Cybernetics* 4.2(2007):100-107.

[21] Dijkstra, E. W. "A note on two problems in connexion with graphs," *Numerische Mathematik* 1.1(1959):269-271.