Full-stack S-DOVS: Autonomous Navigation in Complete Real-World Dynamic Scenarios

Diego Martínez(✉), Luis Riazuelo, and Luis Montano

Engineering Research Institute of Aragon (I3A), University of Zaragoza, Zaragoza, Spain
{diegomartinez,riazuelo,montano}@unizar.es

Abstract. Autonomous navigation in dynamic environments is a nowadays unsolved challenge. Several approaches have been proposed to solve it, but they either have a low success rate, do not consider robot kinodynamic constraints or are not able to navigate through big scenarios where the known map information is needed. In this work, a previously existing planner, the Strategy-based Dynamic Object Velocity Space, S-DOVS, is modified and adapted to be included in a full navigation stack, with a localization system, an obstacle tracker and a global planner. The result is a system that is able to navigate successfully in real-world scenarios, where it may face complex challenges as dynamic obstacles or replanning. The final work is exhaustively tested in simulation and in a ground robot.

Keywords: Autonomous navigation · Dynamic environment · Obstacle avoidance

1 Introduction

Robotic autonomous navigation is a problem that has been addressed in many works. Traditional planners have proven to be able to navigate successfully and efficiently in static scenarios. They use the information received by the robot sensors to compute a path that it must follow. Nevertheless, most of that commonly used planners fail when they face dynamic obstacles, as they do not take their variables into account, as in the scenario shown in Fig. 1.

A dynamic obstacle is an object that does not have null velocity and the robot should avoid. Not only the current position of the obstacle must be taken into account, but also other variables as its velocity or its trajectory. Moreover, the velocity of the obstacle may vary. All of those aspects must be considered. Otherwise, the robot could collide with dynamic obstacles or have suboptimal behaviors.

A planner designed to navigate in dynamic scenarios is the S-DOVS, proposed in [1]. It uses the Dynamic Object Velocity Space (DOVS) to identify in which state the robot is and its relation with the environment. Based on that, it identifies a situation and applies a velocity to the robot according to a predefined strategy. The planner differs from other planners in the fact that it considers the

https://doi.org/10.1007/978-3-031-21062-4_2

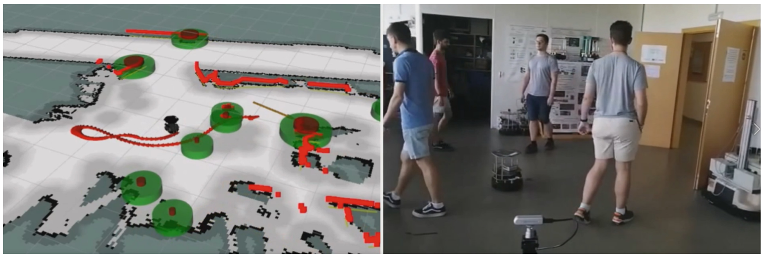


Fig. 1. An autonomous robot navigating in a dynamic scenario.

robot kinodynamic constraints and uses a deep abstraction of the environment, which is the DOVS.

The S-DOVS planner, however, has some limitations. It is able to navigate through two points that are directly connected, but it is not able in isolation to reach a goal in a big known map, where it should navigate through intermediate goals to exploit the map information and connections. It is not able to avoid convex objects or with big walls, or trap situations, where a replanning behavior would be intended. Moreover, it assumes perfect knowledge of the environment, itself and other obstacles.

In this work, we improve and complete the work presented in [1] to include a modified version of the S-DOVS planner in a full navigation stack, obtaining an autonomous robot that may navigate autonomously through real-world scenarios.

2 Background

2.1 Related Work

We can find in the literature a wide range of iterative motion planners. They compute a safe plan to reach the goal, recomputing it reactively if they find any obstacle that interferes with the path. The Dynamic Window Approach (DWA) [2] is a well established and widely used local planner for static environments and non-holonomic robots, which computes the next motion command in the robot velocity space by optimizing a criterion that weights maximum velocities, distance to obstacles, and motion towards the goal. It only contemplates the velocities reachable by the robot within its kinodynamic constraints, but does not consider future collisions with obstacle trajectories. In [3] the global dynamic window approach is also applied for computing high-velocity motions.

Other approaches include the representation of the robot movement as the result of potential forces, as the artificial potential fields presented in [4] or representing the environment as a spring-mass system, as the elastic band [5].

The previous works develop planning methods for static or no highly dynamic environments. In [6–8], planners for dynamic environments are developed. Including dynamic obstacles in the model requires a more complex representation than just using obstacles' positions as if they were static. There is a wide range of methods that use the velocity of the obstacles in the model. Two models have been widely used in the literature. One of them are the *inevitable collision states (ICS)*, introduced in [9], where the dynamics of the system and the obstacles are used to compute future states where a collision is inevitable. The other one is the *Velocity Obstacle (VO)*, introduced in [10]. The VO is defined with the trajectories the dynamic obstacles have in time, computing those that could cause a collision with the robot. The velocities that are not inside the velocity obstacles are available to be chosen by the robot. The model ensures also that the robot dynamics are respected.

In [1], a model to represent the environment dynamism is defined for non-holonomic robots, the Dynamic Object Velocity Space (DOVS), where the robot kinodynamic constraints are included. Using this model, a strategy-based planner is developed, the S-DOVS, which identifies a situation among several defined as a function of the relative robot and moving objects variables. For each situation, the method applies a different action (robot linear and angular velocities), balancing the time to goal and robot safety.

In recent years, learning approaches to solve the problem of motion planning in dynamic environments have gained importance, in works like [11–13] or [14], which also uses the DOVS. They try to use reinforcement learning to solve the limitations of model-based approaches in problems with high complexity and a big number of variables. However, the gap between the light-weight simulations used to train and the real world is huge. The simulators are not able to reproduce exactly the real-world physics and randomness, meaning that the interactions with the environment are biased. They do not consider robot kinodynamics, assuming the robots do not have acceleration constraints, may choose any velocity at any time and they may have maximum linear and angular acceleration at the same time. Moreover, they usually suppose the robot is holonomic, which is not common in real life, or has perfect knowledge of the environment.

2.2 Dynamic Object Velocity Space (DOVS)

The velocity space (VS) is defined as the set of velocities the robot may reach with the constraints of maximum and minimum velocities imposed. Some of the velocities would make the robot collide with another object, while others are safe velocities. The Dynamic Object Velocity Space (DOVS) is addressed in [1]. This model abstracts the dynamic environment, representing safe and unsafe velocities the robot can select every sampling period. The Dynamic Object Velocity (DOV) is defined as the velocities of VS that can lead to a collision in a time horizon. The methods that use this model, may choose the next velocity command in a control period among the safe velocities outside DOV.

DOVS is used to navigate through dynamic scenarios by computing safe motion commands within a time horizon. The robot size is reduced to a point, enlarging the obstacles with the robot radius. Then, for each moving obstacle, the collision band is defined as the area swept by an enlarged obstacle whilst it moves. In each possible pair of angular and linear velocities, the robot motion results in a circular trajectory while the velocities are kept. The intersection of the trajectories with the collision band define the possible collision points of the robot with the obstacle, if at the time in which the robot passes through that point, the obstacle is there too. The maximum and minimum velocities that lead to a collision with every obstacle are computed, obtaining from the sets of safe and unsafe velocities. A dynamic window is used to set the velocities the robot may take in the next step within its kinodynamic constraints.

The DOVS may be graphically represented, as it is shown in Fig. 2. In the image, safe velocities are represented in white, unsafe are dark, the dynamic window is represented with a green rhombus, the velocities that lead to the goal with a magenta line, safe velocities close to the ones that align the robot to the goal with a green line, and the big triangle represent the robot velocity limits corresponding to a differential drive robot.

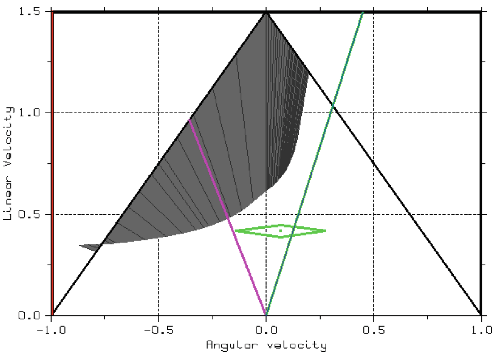


Fig. 2. A graphical representation of the DOVS.

The robot may choose a velocity among a set of only 8 possible velocities: The velocities of the extremes of the rhombus, the current velocity of the robot (center of the rhombus), the intersection of the green line with the rhombus (minimum and maximum velocities that lead to the goal) and the velocity that leads to the goal with the current linear velocity. Those last three velocities are only available if the velocities that lead to the goal are inside the dynamic window. The robot selects an action based on strategies, as explained in [1]. Depending on the situation of the agent and the objects that surround it, a state is identified as one of the predefined situations in a tree. For each recognized situation, a different behavior is designed, so that the robot avoids collisions and reaches the goal, balancing the safety and the maxima robot velocities in every sampling time. There is an extension of this work [14], where the predefined strategies are substituted by a decision-making process using a basic reinforcement learning method based on Q-tables for representing the accumulated reward for each state.

2.3 Contribution

Many of the works using learning based approaches for navigation in dynamic environments are typically trained in unrealistic simulations, while the S-DOVS model-based planner was designed regarding real-world physics and dynamics. Nevertheless, the original S-DOVS planner has some limitations, as it is a local planner to move between goals. It is not integrated with a global planner to navigate in environments with multiple rooms or convex objects due to the absence of a map, and has not a replanning capability in trap situations. It also assumes that the robot and obstacle localization is perfectly known, focusing on the navigation problem. The contributions of this work are:

- Modifications and improvements in the S-DOVS original model.
- A decentralization of the DOVS model, allowing a multi-robot execution, with each robot performing its own computations.
- An adaptation of an obstacle tracker to be used alongside the S-DOVS, getting rid of the perfect knowledge of the environment.
- An inclusion of a global planner that is able to work with the S-DOVS while enhancing its strengths.
- A ROS implementation and adaptation of the model, allowing the execution in both simulation and the real world, respecting robot kinodynamics in both of them.
- Evaluation in simulation and real-world experiments.

3 Approach

The approach taken was designing a whole robotic system to be able to navigate in complete real-world dynamic scenarios by using S-DOVS.

3.1 Decentralization and ROS

The first step of the work was adapting the program to be used directly on a real robot. The initial setup proposed in [1] was purely written in C++, and all the computations were achieved by the main program. The ground truth information was also provided by the simulator, not by any sensor. The initial step was adapting the algorithm to be used with the Robotic Operating System (ROS) [15], an open-source framework that includes a set of libraries and tools for robotic applications. A simple diagram of the ROS aspects used in this work and the way they work is shown in Fig. 3.

Stage [16], which is one of the best simulators integrated in ROS, was used. It is a lightweight simulator that is able to include multiple robots in the same scenario with few computational resources. In addition, it simulates realistically the robots' dynamics and kinematics, as well as their sensors and world physics. The simulator is modified so that collisions are detected and the agents may be teleported to start a new episode without restarting the simulation. Two types of agents were developed. Some of them, active agents, use the S-DOVS

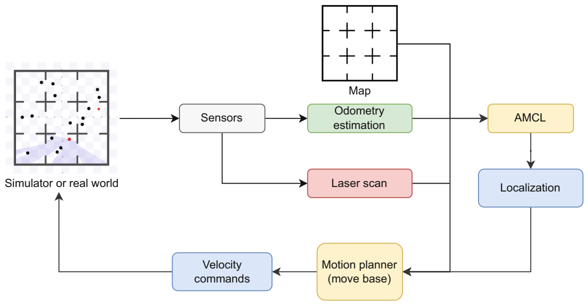
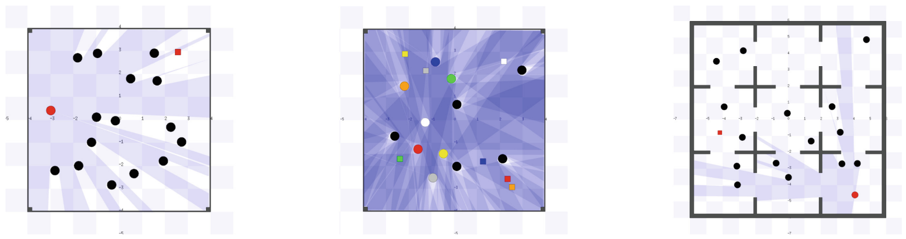


Fig. 3. A simple diagram of ROS. Sensors information is used for localization (AMCL node) and planning (typically move base node).

algorithm for navigation. The others have a preferred linear and angular velocity, which they keep until they find a wall. Then, they turn around and restore their previous velocities. Therefore, passive agents do not avoid collisions and may act as dynamic obstacles. The launching is performed with an automatic script that creates the agents, the environment and launches them. An example of simulations with different number of agents is shown in Fig. 4.



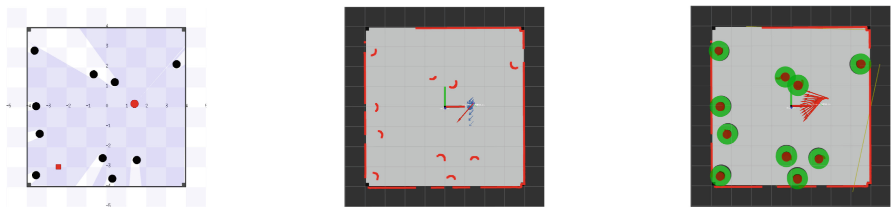
(a) 1 active, 15 passive (b) 7 active, 5 passive (c) 9 rooms scenario

Fig. 4. Example of different scenarios in Stage. Passive agents are in black and active agents (with a laser sensor) in the same color as their goals. (Color figure online)

3.2 Localization

The robot needs to know where it is to be able to navigate in a scenario. In a centralized program and in simulation, the position of the robot is known. Stage does it, and this approach is used for passive agents, which do not perform any computation. Nevertheless, active agents should behave as in the real world, and they should not use that information. As sensors, the robots only have a 2-D LIDAR sensor and the wheels encoders for estimating the odometry.

Having the static map of the scenario and the 2-D laser scans from the onboard rangefinder sensor, the approach applied for localization is using Adaptive Monte Carlo Localization (AMCL) [17], provided in the ROS navigation stack. It uses a high number of particles that are placed in possible robot poses. As more measurements are collected, the particles are updated and sampled in the places where the robot is more likely to be, converging to the robot's current position after a few iterations. Therefore, the robot is localized by using the AMCL, which takes as the input the map, the 2-D laser scan, the initial position and the encoders' measurements; and publishes the robot pose in the map frame. A representation of the AMCL in a real execution is shown in Fig. 5(b).



(a) Scenario in Stage

(b) RVIZ with AMCL

(c) RVIZ with obstacles.

Fig. 5. The visualization of Stage and what the robot senses in RVIZ; with AMCL pose (red), particles (blue) and laser scans in (b); and obstacles and odometry in (c). (Color figure online)

3.3 Global Planner

The S-DOVS local planner is able to navigate in dynamic environments, where the initial position and the goal are connected with a straight line, as the other planners for dynamic environments in the state-of-the-art. Nevertheless, to navigate in the real world, using the information extracted from a known map is helpful. The robot could find a large wall that should surround or could be trapped by two walls or by other obstacles temporally stopped, problems that could be sorted out by a global planner. Moreover, the original static map may be dynamically updated, so that the robot could add sensed static obstacles and take them into account in new plans. For instance, the robot may try to enter a room through a door, find out that the door is closed, and replan to find a path through a different entrance.

The typical approach in robot navigation is having a global planner computing a dense path of points and the local planner in charge of following them. However, one of the key aspects of the S-DOVS is that it takes only one point as a goal that could be far from the initial pose, and plans the trajectory in advance with the estimated velocities of the obstacles. Thus, the S-DOVS should receive points that are connected in the map and far enough from each other, so that it has the freedom to maneuver properly.

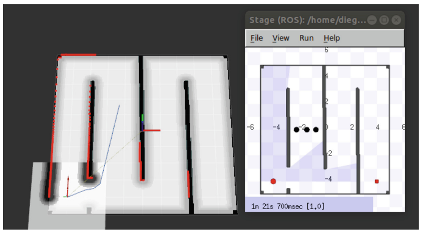
In this work, the A* implementation of the ROS standard move base has been used as a preliminary global planner, to compute an initial global path. Moreover, the map is kept updated so that the global cost map takes into account newly seen static obstacles, and the A* algorithm plans accordingly. The inflation ratio of the cost map is set to the double of the robot size, so that the S-DOVS has enough space to choose its own direction.

From the initial global plan precomputed by the A*, our global planner takes one point per square meter, ending up in a plan with separated points. Intermediate points in a straight line of the plan are undesirable, as the S-DOVS could have suboptimal performance if trying to reach all of them instead of computing the velocities to reach the last one. To remove them, for each three points, if they could be considered to be in the same line, the point of the middle is removed. To check it, we test whether the slope of the line that passes through the first and the second points is similar than the slope of the line that passes through the first and the third points. This is done with the following equation:

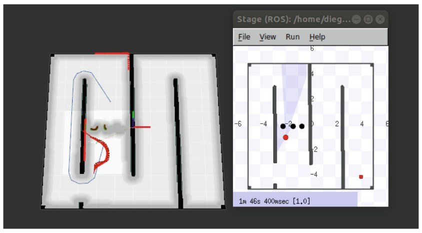
$$\frac{y_1 - y_2}{x_1 - x_2} \approx \frac{y_1 - y_3}{x_1 - x_3} \implies |(y_1 - y_2)(x_1 - x_3) - (y_1 - y_3)(x_1 - x_2)| < \epsilon, \quad (1)$$

where ϵ is a constant set to 0.05 and the points coordinates are (x_i, y_i).

When the robot gets close to an intermediate goal, but before it is reached, the S-DOVS is fed with the next goal of the path, so that the robot does not stop in the points of the path and moves fluently. In addition, when the robot gets too far from the plan computed by the global planner or stops because it is in a trap situation, a new plan is recomputed. With this behavior, the robot follows points that are adapted to the trajectory of the S-DOVS and is able to replan when the map is updated with new static obstacles that block its way. An example of this replanning behavior is shown in Fig. 6.



(a) The original path



(b) The path after replanning

Fig. 6. A visualization of the global planner behavior. The computed path is shown in blue. The robot tries to reach the goal with the shortest way, but replans a new path after updating the map with the obstacles. (Color figure online)

3.4 Obstacle Tracker

In a realistic scenario, each active agent must gather the information of the other obstacles from the sensor measurements. The DOVS model uses the current obstacles' position, radius and velocity to predict their trajectory and their collision band; which must be computed using the LIDAR sensor.

To solve this problem, the work presented in [18] is used. It first takes the laser measurements and group them to form circles or segments. The obstacles' center and their size (radius or length, respectively) are stored. Then, an Extended Kalman Filter is used to track the measurements in time and check whether new obstacles discovered belong to previous measurement or not. The size, position and linear velocity of the obstacles is tracked through the time. Moreover, if an obstacle is no longer seen, it keeps its track a few steps in case it is seen again.

That work is extended, as the orientation and angular velocity of the obstacles are needed too. The orientation is computed by using the direction of the obstacle trajectory, while the angular velocity is computed by tracking the change in the orientation with a high frequency. This is performed to be aware of sudden changes in the obstacles' motion. Nevertheless, spurious measurements or occlusions lead to wrong estimations. Thus, a median filter is used to remove wrong angular velocity estimations. A visualization of the obstacle tracker in a real execution is shown in Fig. 5(c).

3.5 Modifications in S-DOVS

One of the main contributions to the original model is the inclusion of segment obstacles. In the original version, all the obstacles were treated as circles, defined by their position, velocity and radius, and walls would be treated as a group of circles (inaccurate and curvy). The solution designed was introducing segment obstacles detected by the obstacle tracker to the model. The way to introduce the segments in the DOVS was by saving the points of the extremes. If a trajectory intersects with a point that is between the extremes of the segment, the velocity that leads to that trajectory is forbidden (any velocity in an early implementation, changed as shown later).

Strategies that make the agent choose velocities that lead to trapping situations or situation with small margin to react have been removed, as well as some other risky situations, such as trying to accelerate when there are no free velocities to try to find any (the new behavior is braking to avoid running over pedestrians).

Another change made is a modification in the calculations of the forbidden velocities for static obstacles. An obstacle is considered static if it is a segment or its velocity is smaller than 0.1 m/s (in the original model, the minimum velocity to be static was 10^{-5} m/s, but in the real world estimation errors may be found). The original model sets that any trajectory that collides with an obstacle in any time is forbidden. However, that could be too conservative for static obstacles, which may be easily avoided. In that case, the trajectories are only considered forbidden if the distance following that trajectory is bigger than the distance necessary to make the robot brake, multiplied by a safety factor. That distance is defined with the following formula:

$$brake\_distance = \frac{v^2}{2 * \max(\Delta v)} \tag{2}$$

4 Experimental Results

The videos of all the experiments may be accessed in the following link: https://youtu.be/fGHZiMt-Yao.

4.1 Simulation Experiments

In the simulation experiments, the effectiveness of the global planner is tested. The global planner should help the S-DOVS to avoid collisions by using the previous static map information, and also by adding new static obstacles to the costmap that are avoided with the global plan.

Two different kinds of experiments are performed. The first ones use a simple 4 m x 4 m room as the scenario. In that scenario, static and dynamic obstacles are randomly placed, whose velocities are random too. The robot starts in a random position and has to navigate to a goal. A set of 200 scenarios are generated for any number of obstacles in the range 1 to 15. The robot ties to complete the same scenario both with only the S-DOVS and with the global planner too. With these experiments, we want to prove that the global planner not only does not worsen the performance, but also improves it. Theoretically, in that kind of scenario, the global planner would not be needed. Nevertheless, the addition of newly seen static obstacles to the costmap could help.

The other kind of experiments use the same structure as the previous one. The only difference is that the map changes. Instead of only one room, 9 rooms are included, as in Fig. 4. In this kind of scenario, the S-DOVS planner will face convex obstacles that could not avoid.

The results obtained in the experiments are shown in Table 1. The robot is able to reach the goal in more episodes when it uses the complete navigation stack, even on the map with only one room, due to the fact that obstacles could remain static in the way to the goal. The global planner adds them to the cost map, helping the S-DOVS. On the 9-room map, the benefits are clear. The S-DOVS needs to reactively know how to navigate through the rooms. The intermediate goals sent by the global planner makes it know how to avoid properly the walls, which could be interpreted as convex obstacles. The approaches perform similarly in terms of time in the single-room map, and comparably in the multi-room map although the whole system is clearly more safe and successful.

Table 1. Difference in success rate $(\frac{\text{number of success complete system}}{\text{number of success S-DOVS only}})$ and time rate $(\frac{\text{mean time complete system}}{\text{mean time S-DOVS only}})$, on a map with a single room and with 9 rooms, for the same 200 random scenarios for each number of agents.

Map	Metric	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Single	Success	1.02	1.12	0.99	1.05	1.02	1.19	1.19	1.02	1.16	1.06	1.23	1.06	1.06	1.29	1.39
	Time	1.02	1.01	0.97	0.99	0.98	1.09	1.05	1.00	1.06	1.08	0.93	1.03	0.89	1.06	1.11
Multi	Success	3.09	2.89	3.41	2.84	3.09	4.16	4.11	3.08	3.67	3.21	3.73	2.86	3.93	3.52	3.53
	Time	1.09	1.32	1.11	1.07	1.21	1.08	1.13	1.07	0.95	1.11	1.25	1.24	1.10	1.10	1.11

4.2 Real-world Experiments

The whole system has been tested on a Turtlebot 2 platform with a NUC with Intel Core i5-6260U CPU and 8 GB of RAM, equipped with a Hokuyo 2D-LIDAR sensor.

The system was analyzed in two experiments. One experiment checked whether the robot could replan properly in the real world. It was sent to a goal and, when the robot tried to follow the closest path, it found a closed door; so it needed to replan and go through an alternative path, as seen in Fig. 7, when the robot has just replanned. The other experiment performed was navigating autonomously in a room with other people acting as dynamic obstacles. In both experiments, the robot was successful, navigating autonomously by using the whole integration of the system. An example of navigation may be seen in the previously shown Fig. 1.

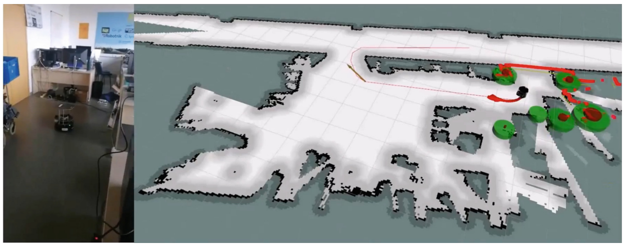


Fig. 7. An image where the robot has just replanned after detecting a closed door, with the global plan shown in red. The global costmap is also shown. (Color figure online)

5 Conclusion

In this work, we develop a complete system that is able to navigate autonomously in dynamic environments, avoiding collisions while minimizing the time to reach the goal. The S-DOVS local planner has been improved and adapted to work in the real world and in realistic scenarios, also considering the robot kinodynamic constraints, which are avoided in many works in the literature. The local planner is integrated in a ROS full navigation stack, a localization node and a global planner. In addition, the proposed navigation stack could be used in other planner intended for dynamic scenarios. The system has been tested in several simulation and real-world experiments, proving a good performance, mainly in more complex scenarios having several rooms.

References

1. Lorente, M.-T., Owen, E., Montano, L.: Model-based Robocentric planning and navigation for dynamic environments. Int. J. Robot. Res. **37**(8), 867–889 (2018)

2. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robot. Autom. Mag. **4**(1), 23–33 (1997)
3. Brock, O., Khatib, O.: High-speed navigation using the global dynamic window approach. In: Proceedings, IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C), vol. 1, pp. 341–346. IEEE (1999)
4. Warren, C.W.: Global path planning using artificial potential fields. In: 1989 IEEE International Conference on Robotics and Automation, pp. 316–317. IEEE Computer Society (1989)
5. Quinlan, S., Khatib, O.: Elastic bands: Connecting path planning and control. In: [1993] Proceedings IEEE International Conference on Robotics and Automation, pp. 802–807. IEEE (1993)
6. Stachniss, C., Burgard, W.: An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 1, pp. 508–513. IEEE (2002)
7. Minguez, J., Montano, L.: Sensor-based robot motion generation in unknown, dynamic and troublesome scenarios. Robot. Auton. Syst. **52**(4), 290–311 (2005)
8. Hsu, D., Kindel, R., Latombe, J.-C., Rock, S.: Randomized kinodynamic motion planning with moving obstacles. Int. J. Robot. Res. **21**(3), 233–255 (2002)
9. Fraichard, T., Asama, H.: Inevitable collision states-a step towards safer robots? Adv. Robot. **18**(10), 1001–1024 (2004)
10. Fiorini, P., Shiller, Z.: Motion planning in dynamic environments using velocity obstacles. Int. J. Robot. Res. **17**(7), 760–772 (1998)
11. Shi, H., Shi, L., Xu, M., Hwang, K.-S.: End-to-end navigation strategy with deep reinforcement learning for mobile robots. IEEE Trans. Industr. Inf. **16**(4), 2393–2402 (2020)
12. Lei, X., Zhang, Z., Dong, P.: Dynamic path planning of unknown environment based on deep reinforcement learning. J. Robot. **2018**, 5781591 (2018)
13. Chen, C., Liu, Y., Kreiss, S., Alahi, A.: Crowd-robot interaction: crowd-aware robot navigation with attention-based deep reinforcement learning. In: 2019 International Conference on Robotics and Automation (ICRA), pp. 6015–6022. IEEE (2019)
14. Mackay, A.K., Riazuelo, L., Montano, L.: RL-DOVS: reinforcement learning for autonomous robot navigation in dynamic environments. Sensors **22**(10), 3847 (2022)
15. Stanford Artificial Intelligence Laboratory: Robotic Operating System. https://www.ros.org
16. Gerkey, B., et al.: The player/stage project: tools for multi-robot and distributed sensor systems. In: Proceedings of the 11th International Conference on Advanced Robotics, vol. 1, pp. 317–323. CiteSeer (2003)
17. Fox, D.: KLD-sampling: adaptive particle filters and mobile robot localization. Adv. Neural Inf. Process. Syst. (NIPS) **14**(1), 26–32 (2001)
18. Przybyła, M.: Detection and tracking of 2D geometric obstacles from LRF data. In: 11th International Workshop on Robot Motion and Control (RoMoCo), vol. 2017, pp. 135–141. IEEE (2017)