Local Trajectory Planning Using Dynamic Random Tree Search

Konstantin Akhmadeev and Bogdan Khomutenko
MCQ-Scan, Lille, France
{k.akhmadeev, b.khomutenko}@mcq-scan.com

***Abstract*—Mobile robots operating in dynamic environments require computationally efficient solutions to the trajectory planning problem that respect kinematic limits and minimize the navigation cost. In this work, we present an approach to plan local trajectories that incrementally advances the robot along a global path, taking into account its kinematic constraints and changes in the environment. The key idea of the presented approach is to combine space discretization and heuristic-based search into a single algorithm. In order to explore the local neighborhood, the algorithm generates a heuristic-driven trajectory tree by iteratively expanding the most promising vertex with randomly sampled control inputs. Then, the robot advances along the best local trajectory and the process is repeated from the new configuration. The approach was successfully applied in a real differential drive robot. Here, we demonstrate the efficacy of our method through tests performed in simulation. The results highlight our approach's suitability for mobile robot navigation in complex dynamic environments.**

***Index Terms*—navigation, local planner, mobile robot, trajectory, sampling-based**

I. Introduction and Related Work

Mobile robots have broad applications in industry and retail. They allow to automate such tasks as delivery, data acquisition, monitoring, maintenance and so on, also being able to carry out such activities in hazardous areas where humans cannot operate.

Fitness of a mobile robot for a certain application primarily hinges on its ability to navigate in the corresponding environment which can be densely cluttered and/or dynamic. The robot should perform the navigation tasks in a safe and cost-efficient manner, e.g minimize the execution time, keep a safe and reasonable distance from obstacles, avoid obstruction of humans' or other robots' paths and exhibit smooth motion to reduce wear and energy consumption.

The task of planning a trajectory that would bring the robot to the goal is, thus, a problem that is widely faced in mobile robotics. There exists a broad variety of approaches to this problem [1], [2]. Some, such as A^* and its variations, provide an optimal path, which is a sequence of robot configurations between its current position and the goal that are collision-free.

However, having only a path is inconvenient for navigation since a path doesn't describe what control inputs should be applied to the robot to perform the task. It doesn't take into account the robot's dynamic and kinematic constraints and doesn't specify the timing of the motion.

In contrast, there are approaches that are capable of computing either a *trajectory* (a time-stamped path) or a direct control input for the actuators. Such methods take into account the kinematic and dynamic models of the robot to be able to satisfy necessary constraints and conditions.

A large class of such approaches is based on the Rapidly Exploring Random Trees (RRT) with introduced kinodynamic constraints [3], [4]. Their advantage, other than the mentioned ability to provide feasible trajectories rather than a path, is a guarantee of convergence under certain conditions.

However, these approaches require computing feasible trajectories between two given configurations, a problem that is also referred to as boundary value problem (BVP) [5]. Some implementations link the two configurations by a straight line [6], Dubin's paths [3], [7], or Reeds-Shepp's curves [8]. Approaches to BVP that are suitable for various kinodynamic constraints (e.g. optimisation-based approaches) can become computationally limiting in cluttered and dynamic environments, which is critical in real-time applications.

Slightly different approach is presented in [9], which samples the configuration space and connects the configurations with B-splines. Application of kinematic and dynamic constraints is ensured by subsequent time warping of the spline trajectory.

Another class of approaches alleviate the need of solving the BVP by directly sampling the control inputs instead of the robot configurations [5], [10], [11]. Resulting trajectories can be obtained by integrating the control inputs using the kinematic model of the robot, which is in general a less costly procedure than solving a BVP. Moreover, the sampling procedure can be designed to guarantee that the control input is feasible, e.g. that velocity or acceleration limits are respected.

However, it is noted in [5] that methods based on control input sampling provide uneven exploration of the configuration space in presence of local minima and are incapable of rerouting existing paths via new less costly vertices. To address these issues, authors of [5] propose to discretize the configuration space and work with a grid instead of a continuous space. This way, local minima can be avoided by exhaustive search, and configurations that find themselves in the same cell can be merged, to make rerouting possible. A

This research has been supported by BPI France, the French government, the Eurostars programme and the European Union.

Notation reference

v_i	i-th vertex of the control tree
x_0	initial configuration of the robot
x_i	configuration of the robot in vertex i, e.g $(\mathrm{x}, \mathrm{y}, \theta)$
t_i	time of arrival to vertex i
u_0	initial velocity of the robot
u_i	control input vector at time t_i, e.g (v, ω)
d_i	duration of control u_i
U_i	sequence $\{u_0, \ldots, u_{\mathrm{parent(parent(i))}}, u_{\mathrm{parent(i)}}, u_i\}$
$\mathbb{U}$	set of valid control inputs
$\phi_i(t)$	continuous control segment between $u_{\mathrm{parent(i)}}$ and u_i, $t \in [t_{\mathrm{parent(i)}}, t_i)$
$\Phi_i(t)$	continuous control between root and u_i; $\phi_0 \circ \cdots \circ \phi_i$, where $\circ$ represents concatenation
$s_i(t)$	trajectory from vertex $parent(i)$ to vertex i, results from integration of $u_i(t)$ starting from x_i
$S_i(t)$	$s_0 \circ \cdots \circ s_i$; trajectory from the root to vertex i
$R(x, u)$	kinematic model of the robot, s.t. $s_i = R(x_i, u_i)$
$M(s)$	indicator function such that $M(s_i) = 1$ if any configuration on s_i is in collision, and 0 otherwise
H_i, C_i	heuristic and cost in vertex i
G	global plan

slightly different approach is presented in [12], which refrains from introducing a grid, and instead promotes the exploration into less densely sampled areas.

When navigating in dynamic environments, it is not always beneficial to plan the trajectory from the start until the very goal. As long as the robot advances through the environment, new obstacles can be discovered, while existing ones may move, thus rendering the previously calculated trajectory obsolete. A more practical approach is to first calculate a rough path towards the goal using a classic planning algorithm such as A^* [13], and then gradually advance along it, performing the trajectory planning only with respect to immediate surroundings of the robot. In fact, it is a common paradigm to dedicate a path planner to produce a *global plan* which is then used as a guide for a *local planner* that finds an optimal trajectory within a limited horizon. A prominent example of this approach is Timed Elastic Band (TEB) local planner [14], a part of ROS navigation stack. In our experience, while generally being able to complete the task, TEB has a few shortcomings that include taking a long time to change the topology of the trajectory when planning around moving objects, occasional loops in the trajectory, as well as backward motion that cannot be fully suppressed.

In this paper, we present an approach to trajectory planning that advances the robot along a predefined global plan and towards the goal, while respecting the kinematic limits and computation time requirements inherent in local trajectory planning. The main contribution of this work is the successful combination of a heuristic-driven dynamically expanding tree search with control input sampling, leading to efficient space exploration and the generation of feasible trajectories.

II. Methods

A. Outline of the algorithm

The approach that we are presenting explores the configuration space by sampling control inputs and integrating them using the robot's kinematic model. By repeating this process

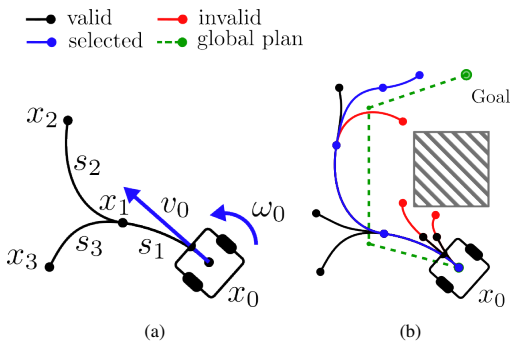


Fig. 1: Schematic examples of control tree; (a) after sampling four control segments, each time starting from the previous vertex; (b) after sampling more control segments, this time starting from arbitrary vertices. Dashed green - global plan, black - valid trajectories, red - trajectories with collisions, blue - best selected trajectory after sampling.

iteratively, it extends the previously obtained controls with newly sampled values. The entirety of the sampled controls form a rooted tree that originates from the current velocity and configuration of the robot.

Let us consider the schematic illustration of our approach presented in Figure 1a. Here, a differential drive robot is in its starting configuration on a 2D plane $x_0 = (\mathrm{x}_0, \mathrm{y}_0, \theta_0)$ with velocity $u_0 = (v_0, \omega_0)$. We will refer to this configuration and velocity as *root vertex*, or v_0. In the first iteration of the algorithm, let us pick a certain continuous control input of duration τ such that $u_1(t) : [0, \tau] \to \mathbb{U}$. Let us integrate it using the kinematic model of the robot, starting from the root configuration, denoting the resulting trajectory s_1 (see Figure 1a). Its end configuration $x_1 = s_1(t_1)$ and velocity $u_1 = \phi_1(t_1)$ reached at $t_1 = t_0 + \tau$, form a new vertex v_1 having root v_0 as its parent.

Let us make two more iterations both times using vertex v_1 as source. This way, we obtain two new control branches $U_2 = u_1 \circ u_2$ and $U_3 = u_1 \circ u_3$ sharing the same initial control segment u_1. Integration of these controls results in trajectories $S_2 = s_1 \circ s_2$ and $S_3 = s_1 \circ s_3$ (see Figure 1a).

By continuing this iterative procedure, we obtain a tree of control inputs (see Figure 1b). In each of its vertices, the tree holds computed values of the associated control input, the end configuration, and the time of arrival. At every iteration, we expand the vertex that currently has the best score, which in its turn is based on the values of heuristic, cost, and other auxiliary components (see II-C for details).

The search stops when a fixed number of iterations is reached. Then, the best trajectory according to a certain scoring criteria is selected as the output of the algorithm. The outline of the described procedure is provided in pseudocode listing 1. The following sections will provide a detailed description of each step of the algorithm.

B. Control sampling and integration

At each iteration of the algorithm, a vertex v_i having the best score is picked as a source for tree expansion (see line 4 in Algorithm 1). Let us denote the control sequence leading to this vertex as U_i. Then, a new control input u_j is sampled, from which a child vertex v_j is instantiated. In general, the

Algorithm 1 *TreeSearch*

Input: Starting configuration x_0 and velocity u_0, collision map M, kinematic model R
initialize priority queue with the root vertex
for $n = 1$ to N **do** ▷ N is max number of iterations
  pop vertex v_i from priority queue
  create a new vertex v_j such that $\mathrm{parent}(j) = i$
  sample control u_j for v_j (see II-B)
  compute interpolated control ϕ_j
  integrate trajectory s_j from v_i to v_j
  if (trajectory s_j introduces collision) **then**
    discard v_j
    continue
  end if
  insert v_j to priority queue (see II-C and II-D)
  re-insert v_i to priority queue with updated score (II-D)
end for
select best vertex $\mathrm{v}_{\mathrm{best}}$ in the tree: (see II-E) **return** $\Phi_{\mathrm{best}}(t)$, $S_{\mathrm{best}}(t)$

presented approach is agnostic to a concrete choice of control sampling strategy. Here, for reference, we describe the one we used in our application.

First, we set the duration of the generated control τ. This value, in combination with the distributions used for control sampling, sets the trade-off between the expansiveness of exploration and the spacial resolution of the resulting tree. In general, τ can be adjusted dynamically, although here we will consider it fixed to simplify the notation.

Then, we sample individual components v_j, and ω_j of vector u_j independently from a uniform distribution (example for linear velocity component v_j):

$$v_j \sim U(v_{\min}, v_{\max}) \tag{1}$$

where v_{min} and v_{max} are defined by kinematic limits of the robot, i.e maximal velocity and acceleration:

$$\begin{aligned} v_{\min} &= \max(v^-_{\lim},\ v_p - a^-_{\lim}\tau) \\ v_{\max} &= \min(v^+_{\lim},\ v_p + a^+_{\lim}\tau) \end{aligned} \tag{2}$$

where τ is duration of the sampled control, v_p is the velocity in parent vertex, $v^-_{\lim}$ and $v^+_{\lim}$ are respectively the minimum and maximum allowed values for linear velocity v; $a^+_{\lim}$ and $a^-_{\lim}$ are respectively the limit acceleration and braking. Expressions for angular velocity component ω are similar to these. One can check for other constraints, such as normal acceleration, and adjust the sampling respectively.

After a new control input u_j is sampled, corresponding trajectory from the parent vertex v_i to v_j is integrated. For that, a continuous control input $\phi_j(t)$, $t \in [t_i, t_j)$, $t_j = t_i + \tau$ is computed by linear interpolation between u_i and u_j. Then, trajectory s_j is calculated by integrating the continuous control $\phi_j(t)$ starting from x_i using the kinematic model of the robot. If s_j is collision-free, vertex v_j is added to the tree.

C. Cost and heuristic functions

Score of the inserted vertex is calculated as a sum of its cost and heuristic values. For the cost of vertex, we use its time of arrival, i.e. duration of the trajectory leading from the root vertex towards this node. We note that since all vertices are associated with control inputs of known duration, this resolves to accumulating the durations of control of the entire parents sequence up to the root:

$$C(\mathrm{v}_j) = d_j + \sum_{p \in \mathrm{parents}(\mathrm{v_j})} d_p + \inf \cdot M(s_j) \tag{3}$$

where d_i is the duration of vertex v_i's control segment u_i, $M(s)$ is an indicator function such that $M(s) = 0$ if trajectory s is collision-free and $M(s) = 1$ otherwise.

Our heuristic score, given that we seek to promote advancement along a global plan, consists of several terms:

$$H(x) = w_L L(x, G) + w_D D(x, G) + w_R R(x, G) \tag{4}$$

where

- $L(x, G)$ is the length of the remainder of the global plan from the closest point up to the goal;
- $D(x, G)$ is the distance from configuration x to the closest point on the global plan P;
- $R(x, G)$ is the error between the orientation of the robot in x and the tangent vector of the global plan at the closest point.

Weights w_L, w_D and w_R can be adjusted to promote faster minimization of certain components. For example, increasing the w_L will prioritize sampling from the configurations that have advanced along the plan further than the other ones. Large values of w_D and w_R will promote trajectories that lie closer to the global plan and are oriented along it.

Tuning these weights can be started by setting w_L to 1.0 as an anchor value. Then, w_D is set to be slightly larger, to promote approaching the plan over moving in parallel to it when the two options would be tied otherwise. Finally, w_R should take a smaller value, so that, when approaching the global plan, the planner is not discouraged from turning towards the plan rather than aligning with it.

D. Density penalization and vertex score

In cases when an obstacle occupies the straight line that connects the robot with the goal, classic heuristic scores, such as euclidean distance, will first direct the search towards the obstacle, and start to explore the potential bypasses only after making an exhaustive search in the obstacle's vicinity. In A^* and related algorithms, an exhaustive search is possible since the configuration space is discrete. However, our approach uses continuous configuration space, and thus requires another mechanism for escaping from local minima of heuristic function.

To address this problem, we have designed an additional component to the vertex score that penalizes vertices proportionally to their local density, an approach similar to the one used in [12]. This way, as the local minimum area is getting more and more densely sampled, the search starts to

prioritize exploration into other regions. Density penalty has the following form:

$$D(x_j) = \sum_{i \in \text{Tree}} \frac{1}{1 + \left[\frac{\|x_j - x_i\|}{\sigma_s}\right]^2} \tag{5}$$

where and σ_s is a scaling factor.

To further improve the algorithm's ability to avoid local minima, we introduce a hard limit on the number of children a vertex may have. Once a vertex reaches this limit, it doesn't get selected for extension anymore.

To promote exploration into the width rather than the depth of the tree, we also impose a hard limit on the depth a vertex may have. This way, the algorithm does not tend to extend too far from the current configuration. Which is of advantage since long plans would have greater probability of being invalidated in the near future by a dynamic obstacle.

If both of the described limit conditions are met, the vertex is added to the queue with score:

$$P_j = C(\mathrm{v}_j) + H(x_j) + D(x_j) \tag{6}$$

where $D(x_j)$ denotes density penalty (see 5);

Vertices that haven't met the aforementioned limits or have received an infinite cost due to a collision (see 3) are added to the tree, but are not inserted to the queue, permanently becoming leafs.

E. Selection of best control

The algorithm continues the exploration for a predefined number of iterations. At this point, each vertex v_i of the resulting tree contains a valid control input $\Phi_i(t)$ that drives the robot from the parent configuration $x_{\text{parent}(i)}$ to x_i as well as computed values of cost $C(\mathrm{v}_i)$ and heuristic $H(x_i)$. At this step, we need to select a trajectory that will advance the robot towards the final goal.

In classic path search algorithms, it is required that the goal has been reached during exploration to produce a solution. However, the amount of time available for the search of local trajectory is limited, thus the search may need to be terminated before the exploration can reach the goal. In these conditions, we must make sure that the produced trajectory does not end in a local minimum. This can be achieved by selecting only the *leaves* of the tree as candidates for best trajectory.

Additionally, if weights of the heuristic score in 4 are adjusted so that it is admissible, then the value of cost+heuristic score only increases as we move from the root. Thus, to produce a trajectory that advances the robot along the plan, we must move towards the vertices whose depth from the root exceed a certain limit.

The set of vertices that are eligible for best trajectory selection can be denoted as:

$$E = \{i \mid \text{children}(\mathrm{v}_i) = \varnothing \wedge \text{depth}(\mathrm{v}_i) \geq D\} \tag{7}$$

Among these candidates, we look for a vertex that has the minimal value of weighted cost+heuristic score among a set of eligible vertices E:

$$\underset{i \in E}{\operatorname{argmin}} \left(C(\mathrm{v}_i) + H(x_i)\right) \tag{8}$$

where $C(\mathrm{v}_i)$ and $H(x_i)$ are, respectively, the values of cost and heuristic in vertex i.

F. Control application

After trajectory $S_{\text{best}}(t)$ is found, it is to be executed (e.g. using a closed-loop controller) for a certain amount of time until the next run of the search is completed.

Let us note that the selected trajectory has a significantly larger horizon than the time gap between subsequent runs of the search. To establish an approximate time scale: the trajectory is planned for several seconds while the search is repeated at a frequency of about 10Hz. Thus, the robot will follow the selected trajectory for a relatively short amount of time prior to recalculating the remainder of it.

The search procedure may either be called at a sufficiently high frequency, and/or be triggered whenever the planned trajectory becomes invalid, e.g. due to appearance of a new obstacle that introduces a possible collision.

III. Results

All of the simulated results presented below have been obtained using a laptop equipped with Intel® Core™ i7-10750H CPU 2.60GHz and 16Gb RAM. The presented algorithm is implemented in C++, with no parallelization.

A. Reactivity in dynamic environment

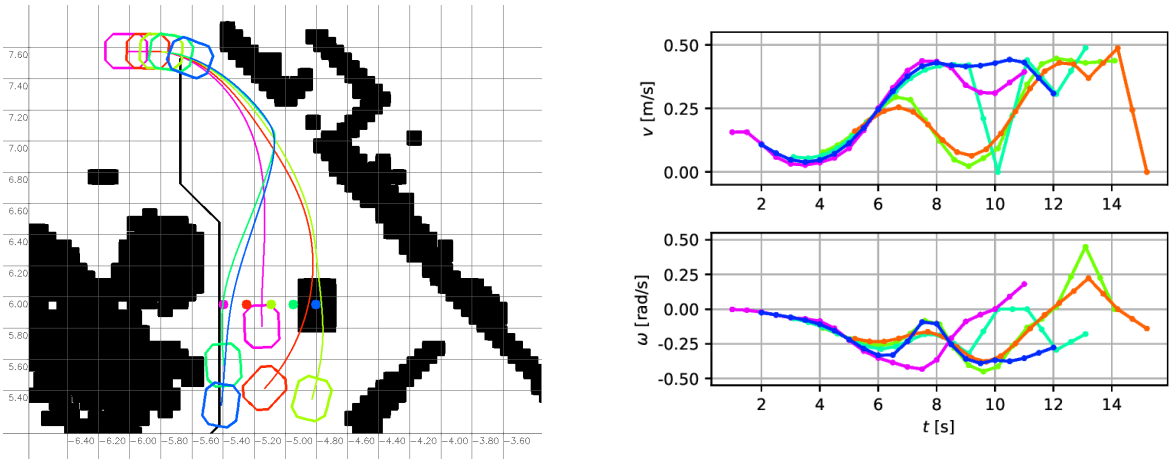
First, we would like to demonstrate the behaviour of the proposed approach in presence of a moving obstacle. In Figure 2a, the robot finds itself on a costmap constructed from a real environment. It is given a global plan (obtained using A^* algorithm) leading from its starting configuration towards the bottom of the image and beyond its scope. We added a simulated dynamic obstacle - a black rectangle that moves across the passage that the robot has to move through.

One can see that the search returns a consistent set of valid trajectories that follow the global plan and avoid the obstacle at its current position. As the obstacle shifts to the right, so does the trajectory (order: violet - red - yellow), until the search considers it best to pass through the other side (order: green - blue). In all of the cases the search made 512 iterations and took less than eight milliseconds, resulting in trajectories planned for 3 to 4 meters ahead.

Figure 2b presents the velocity control inputs corresponding to the trajectories from Figure 2a (colors match). One can see that control inputs are consistent with each other on the first third of the trajectory, becoming dissimilar as the search starts to prefer a different route. All accelerations are within the imposed limits (0.5 $\mathrm{m/s^2}$ and 0.5 $\mathrm{rad/s^2}$ for linear and angular acceleration respectively).

B. Large-scale navigation

To test the proposed approach in larger scale environment, we have simulated a 100×100 m store containing shelves of different sizes arranged in various configurations (see Figure



(a) (b)

Fig. 2: (a) Illustration of trajectory re-planning in the presence of a dynamic obstacle. Black cells - occupied regions of the costmap; solid black line - global plan. Each color corresponds to a different time snapshot (violet - first, blue - last); colored dots - center of the dynamic obstacle; colored solid lines - trajectory calculated by the search algorithm; rounded rectangles - footprint of the robot at the start and at the end of corresponding trajectory.

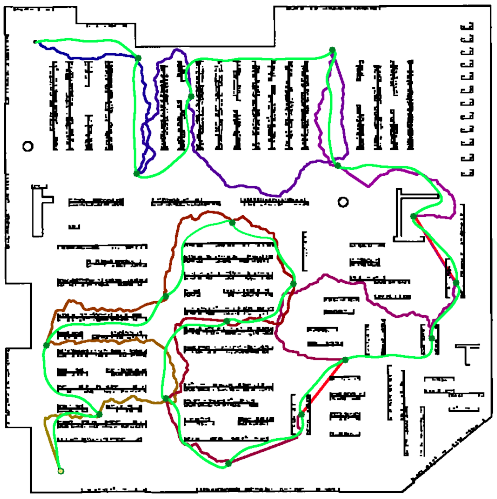


Fig. 3: Trajectories from navigating in a large store-like simulated environment. Dark green dots - intermediate global goals, green solid line - trajectory by our approach, solid blue line transitioning into red - kinodynamic kRRT* (color transition used to visually distinguish overlapping parts of the trajectory), straight solid red lines - tasks that kRRT* failed to compute within the given number of iterations.

3). In that environment, we have set a number of goals to be visited in a specific order.

When using the proposed approach, final trajectory duration was 627 s, while in total 6.3 s were spent in global plan computation by A* and 33.47 s in trajectory computation using the proposed search procedure. The search was invoked at constant time period of 0.1 s, each run took 7.61±0.12ms in average. Once the search returned a trajectory that ends in the goal, the subsequent calls were skipped until this goal was reached. The total length of resulting trajectory is 430 m, robot's velocity limits were set to 1 m/s and 1 rad/s for linear and angular velocities respectively.

To establish a reference, we have ran kinodynamic RRT* (kRRT*) algorithm [3] on the same set of goals. For simplicity, robot's footprint was reduced to a single point, and steering procedure consisted of separate rotation and advancement steps. At each step, maximum limit velocity was used, no acceleration constraint was applied.

TABLE I: Performance comparison for ordered goals task

	Proposed	KRRT
Total time (s)	627	1099
Trajectory length (m)	430	721
Search time (s)	33.47	47.8
A* time (s)	6.3	-
Failed tasks	0	2

TABLE II: Navigation procedures performance

	Total	Per call (ms)
Navigation time	1414.57±106.14 s	-
Trajectory length	1035±81 m	-
A* time	38.23±7.91 s	61.08±56.14
Search time	120.10±19.18 s	7.63±0.14

We have set the number of iterations that kRRT* makes before giving up the task to 25.000 so that its total search time matches the total search time of the proposed approach from the previous example. Search by kRRT* was ran once per each goal from the current configuration of the robot. In total, the simulated time required to complete all goals was 1099 s, out of which kRRT* took 47.9 s. These results are aggregated in Table I.

While given a more complete set of constraints and having made a similar or smaller amount of computations, the proposed approach has produced significantly shorter trajectories (see solid green line in Figure 3) than kRRT* (solid blue). The latter has also failed to compute three tasks out of nineteen within the given amount of iterations.

In order to estimate how the proposed approach scales when given goals at longer distances, we introduce larger

gaps between the goals from 3 by randomizing the order in which they are visited. The results averaged over one hundred runs are presented in Table II.

Visiting the points from 3 in random order increases the distance between the start and goal configurations, thus significantly increasing the global planning time (see A^* in Table II, compared to 6.3 s in previous example).

IV. Discussion

Presented approach has been tested on a real differential-drive mobile platform and provided a reliable and safe motion in a dynamic office environment. We would like to discuss the questions that have arisen during practical implementation of the proposed approach.

A. Trajectory optimization

Although the computed trajectories respect the kinematic limits and are obstacle-free, they result from random sampling and are not necessarily optimal with respect to other criteria. As an example, in Figure 2a, the last trajectories (green and blue) could keep greater clearance from the static obstacle on the left. However, for the search, any trajectory that is further from that obstacle would also become longer than the one it has already found and therefore would have worse cost+heuristic score.

To address this issue in our implementation of the planner deployed on a real robot, we apply an additional optimization procedure to the resulting control $\Phi_{\mathrm{best}}(t)$. It resamples the control nodes U at higher frequency and adjusts their values to optimize a certain number of criteria, such as: clearance from the obstacles, smoothness of control and total duration of motion. Additionally, once the trajectory reaches the end of the plan, the optimisation draws its end closer to the goal and establishes gradual braking. We plan to present this optimization procedure in a separate follow-up paper.

B. Narrow passages

Another common problem arising from random control sampling [5] is the struggle with finding a trajectory through a narrow passage. As the clearance from the obstacles on both sides of the robot approaches zero, so does the probability of sampling a valid trajectory that drives between them.

In order to allow for planning through narrow passages, we use an eroded footprint for the search procedure. After a valid trajectory has been sampled this way, we restore the robot's original size and apply the optimization procedure to adjust it to the passage. If the optimization fails to fit the original footprint through the passage, the planner discards the current results and restarts the search from scratch.

V. Conclusion

In this paper, we have described our novel practical approach to local trajectory planning in dynamic environments. It calculates a valid trajectory that respects the robot's kinematic constraints, at a rate that is sufficiently fast for common navigation tasks in mobile robots. Additional global planning procedure, required for the proposed method to function, does not add a significant overhead in medium-sized environments, however specific measures should be taken in case of large one, e.g. navigating between connected sub-maps.

The aspects of practical application of the proposed method with a real robot have also been discussed. While the method can be generalized to a wide area of applications (e.g., different robot models, multidimensional maps, different control laws), additional processing of the resulting trajectory may be required. The proposed approach is currently being used in real robots deployed in warehouse and office environments, demonstrating reliable and fast trajectory planning.

References

[1] J. R. Sánchez-Ibáñez, C. J. Pérez-del Pulgar, and A. García-Cerezo, "Path planning for autonomous mobile robots: a review," *Sensors*, vol. 21, no. 23, 2021.

[2] H.-y. Zhang, W.-m. Lin, and A.-x. Chen, "Path planning for the mobile robot: a review," *Symmetry*, vol. 10, no. 10, 2018.

[3] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 7681–7687.

[4] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 1, 1999, pp. 473–479 vol.1.

[5] D. D. Dunlap, C. V. Caldwell, and E. G. Collins, "Nonlinear model predictive control using sampling and goal-directed optimization," in *2010 IEEE International Conference on Control Applications*, 2010, pp. 1349–1356.

[6] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Robotics: Science and Systems VI*. The MIT Press, 08 2011. [Online]. Available: https://doi.org/10.7551/mitpress/9123.003.0038

[7] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[8] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, vol. 145, pp. 367–393, 1990.

[9] M. Haddad, T. Chettibi, S. Hanchi, and H. Lehtihet, "A random-profile approach for trajectory planning of wheeled mobile robots," *European Journal of Mechanics - A/Solids*, vol. 26, no. 3, pp. 519–540, 2007.

[10] L.-L. Wang and L.-X. Pan, "Research on SBMPC algorithm for path planning of rescue and detection robot," *Discrete Dynamics in Nature and Society*, vol. 2020, p. 7821942, Nov. 2020, publisher: Hindawi.

[11] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[12] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, Mar. 2002, publisher: SAGE Publications Ltd STM.

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

[14] C. Rösmann, F. Hoffmann, and T. Bertram, "Kinodynamic trajectory optimization and control for car-like robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 5681–5686.