Monte Carlo Tree Search with Velocity Obstacles for safe and efficient motion planning in dynamic environments

Lorenzo Bonanni*
University of Verona
Verona, Italy
lorenzo.bonanni@univr.it

Daniele Meli*
University of Verona
Verona, Italy
daniele.meli@univr.it

Alberto Castellini
University of Verona
Verona, Italy
alberto.castellini@univr.it

Alessandro Farinelli
University of Verona
Verona, Italy
alessandro.farinelli@univr.it

ABSTRACT

Online motion planning is a challenging problem for intelligent robots moving in dense environments with dynamic obstacles, e.g., crowds. In this work, we propose a novel approach for optimal and safe online motion planning with minimal information about dynamic obstacles. Specifically, our approach requires only the current position of the obstacles and their maximum speed, but it does not need any information about their exact trajectories or dynamic model. The proposed methodology combines Monte Carlo Tree Search (MCTS), for online optimal planning via model simulations, with Velocity Obstacles (VO), for obstacle avoidance. We perform experiments in a cluttered simulated environment with walls, and up to 40 dynamic obstacles moving with random velocities and directions. With an ablation study, we show the key contribution of VO in scaling up the efficiency of MCTS, selecting the safest and most rewarding actions in the tree of simulations. Moreover, we show the superiority of our methodology with respect to state-of-the-art planners, including Non-linear Model Predictive Control (NMPC), in terms of improved collision rate, computational and task performance.

KEYWORDS

Motion Planning, Markov Decision Processes, Monte Carlo Tree Search, Velocity Obstacles

ACM Reference Format:
Lorenzo Bonanni*, Daniele Meli*, Alberto Castellini, and Alessandro Farinelli. 2025. Monte Carlo Tree Search with Velocity Obstacles for safe and efficient motion planning in dynamic environments. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 10 pages.

1 INTRODUCTION

Motion planning for mobile robots is an important and widely studied research area. When robots move in a (possibly uncertain) environment in the presence of dynamic obstacles, e.g., other agents or people, they must balance trajectory optimality towards the goal and risk of collision. Fast real-time trajectory computation is a key feature in dynamic environments to guarantee prompt response and adaptation to changes in the environment [39].

Reactive planning methods, e.g., Velocity Obstacles (VO) [20] and Artificial Potential Fields [52] consider a single motion command per time step, but they can get stuck in local minima when maps are complex [19]. On the other hand, *look-ahead planning methods*, as Non-linear Model Predictive Control (NMPC) [32] and tree-based search [46], are more robust since they optimize the trajectory over a time horizon. However, they are computationally demanding in dynamic environments, where re-planning is often needed. Furthermore, these methods often require precise knowledge about the trajectories of dynamic obstacles [51], which is often unavailable or uncertain in real-world domains, especially when dealing with agents exhibiting heterogeneous behaviors (e.g., humans [25]). Deep Reinforcement Learning (DRL) has recently become popular for motion planning in complex dynamic environments; however, it often struggles to generalize beyond the training scenario, offering no guarantee about safe collision avoidance. Additionally, its effective implementation requires extensive training data [1].

In this paper, we propose a novel approach to online robotic motion planning in dense and dynamic and partially unknown environments, combining look-ahead and reactive planning. Unlike other methods assuming partially known obstacle trajectories [31, 51], our approach only requires knowledge about the instantaneous obstacle locations, which can be obtained from standard sensors (e.g., LIDARs and cameras), and an upper bound on their maximum velocity (typically available, e.g., from social models [27] or other robots' specifications). We define the problem as a Markov Decision Process (MDP) and solve it using Monte Carlo Tree Search (MCTS). MCTS often fails to scale to large action spaces [7, 8, 11, 15], which limits its applicability to fine-control mobile robots' velocity. For this reason, we combine MCTS with the VO paradigm, which prunes unsafe actions (i.e., leading to collisions) from the search space during MCTS simulations, reducing the action search space. If the positions and maximum velocities of obstacles are known[1], the integration of the two approaches guarantees that the agent always picks *safe (i.e., not colliding) and optimal actions*, considering a sufficiently small time-step Crucially, using the proposed approach, the number of simulations required by MCTS is significantly reduced,

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

[1] In case of uncertainties, upper bounds can be used.

fostering the deployment of the approach to real robotic platforms, which often have limited computational resources.

We validate our methodology in a simulated 10×10 m map (inspired from [25]) containing up to 40 randomly moving obstacles with a diameter of 0.2 m each. Our results show that VO allows to greatly improve the performance of MCTS, achieving higher cumulative reward and success rate (i.e., collision-free goal reaching) even with very few simulations (up to 10 simulations per MCTS step, corresponding to a planning time of $\approx$0.02 s). Furthermore, the proposed approach achieves superior cumulative reward with fewer collisions with respect to competitor motion planners, including the classical Dynamic Window Approach (DWA) [33] and NMPC [32], an established methodology for optimal motion planning [51] which is considerably more computationally demanding.

In summary, the contributions of this work are the following: First we propose a novel methodology for online motion planning in partially unknown cluttered dynamic environments, *requiring only the knowledge of the current position of obstacles*, rather than their velocities or full trajectories; Second we discuss the assumptions to *guarantee safe collision avoidance* with VO action pruning in MCTS; we scale up the efficiency of MCTS to large action spaces (up to 60 actions) required for smooth trajectory generation in real robotic applications, introducing a VO-based methodology to prune unsafe actions and reduce required simulations; Third we empirically demonstrate the efficacy of our methodology in environments with dense randomly moving obstacles, compared to state-of-the-art online planners (NMPC and DWA), and perform an ablation study to highlight benefits of VO in MCTS.

2 RELATED WORKS

The problem of motion planning for mobile robots has been extensively studied in scientific literature [3, 16]. Nonetheless, this is still an open research area, given the demonstrated intractability of the problem in generic dynamic environments [30]. We classify motion planning algorithms into three main categories: reactive planners, look-ahead planners, and learning-based planners. *Reactive planners* compute only the next safe (i.e., collision-free) robot command, given the current configuration of the environmental map. Since they consider only the instantaneous situation, reactive planners are computationally efficient, hence the trajectory can be adapted at run time in the presence of dynamic obstacles and multiple moving agents. Main examples include Artificial Potential Fields (APF) [23, 24, 54] and Velocity Obstacles (VO) [20, 40]. A known issue with reactive planners, such as the APF method, is that they can get stuck in local minima [19] in case of specific configurations of obstacles and goals. This typically requires ad-hoc modifications of the standard reactive planning approach [41]. In the VO paradigm, a slight perturbation of the reactive planner may help escape local minima[2]; however, in cluttered environments this may badly affect the performance of the agent. In complex maps, *look-ahead planners* are more suitable to find a feasible path towards the goal, since they optimize the trajectory over a time horizon. Most popular examples include planners based on graph search to optimize the trajectory over a discretization of the environmental map, including Rapidly-exploring Random Trees (RRT) [34] and A* [18]. Other look-ahead solutions include time-optimal planning [21], Dynamic Window Approach (DWA) [22], Non-linear Model Predictive Control (NMPC) [32, 42] and MCTS [12, 25, 49]. Graph-based planners typically fail to scale in cluttered environments. In such settings, the environment must be represented with very fine-grained discretization. This results in a prohibitively large search space. Furthermore, these approaches often require substantial computational resources in dynamic scenarios, where frequent re-planning is necessary. For this reason, look-ahead planners typically require prior knowledge of the trajectories of moving obstacles [51], which however requires unrealistic perfect communication or perception capabilities. In the context of crowd navigation, recent works [25] integrates a probabilistic model of human behaviour into a partially observable MDP for online safe navigation. However, planning over a partially observable state space introduces additional computational complexity; moreover, different obstacles may require different trajectory models, which can only be approximated via learning algorithms [31], without any guarantee of safe collision avoidance when deployed in the real world. Recently, *learning-based* planners exploiting DRL have been proposed to solve the motion planning problem in complex dynamic environments [4, 55]. In [44], authors combined MCTS with a neural network trained from expert demonstrations for non-cooperative robot navigation. However, these methods typically require large training datasets and significant computational power for the training phase. Moreover, they do not provide guarantees in inference on generic maps, thus requiring specific approaches, e.g., posterior formal verification, for safe deployment on real robots [1]. In fact, combining the goal reward with the cost related to obstacle avoidance requires accurate multi-objective reward shaping, which may be sub-optimal and provides no guarantee when applied to real robotic systems [26]. To overcome the limitations of existing planners, we combine the benefits of look-ahead planning with MCTS and reactive planning with VO. It is important to note that our approach operates within a context where the robot lacks knowledge of obstacle trajectories and behaviors, precluding direct comparisons with some state-of-the-art techniques, such as [51] (where obstacle trajectories are assumed to be known) and [25] (where a pedestrian model is assumed to be known). Instead, our assumptions emulate the realistic scenario where the robot moves in partially unknown environments in the presence of heterogenous obstacles (e.g., other robots or humans) with an upper bound on their maximum velocity, and can only rely on its sensors (e.g., LIDARs or cameras) to estimate the positions of other entities. In addition, MCTS performs online planning, hence our method does not require learning and it cannot be compared to DRL approaches which require offline training. We use VO to prune the robot's unsafe actions (i.e., command velocities). In this way, our methodology can scale up to larger action spaces than the state-of-the-art [47] (up to 60 velocity actions in our case), required for application in realistic robotic settings. Moreover, our algorithm is successful and efficient even in the presence of cluttered dynamic environments, where approaches based on map discretization typically fail [2]. Finally, differently from existing methods for motion planning combining MCTS with efficient heuristics [44], our methodology does not require offline training or the availability of expert demonstrations.

[2]Example implementation available at https://gamma.cs.unc.edu/RVO2/

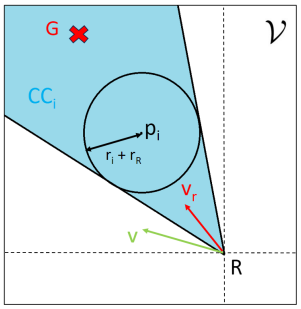


Figure 1: The velocity space for a robot moving in an environment with one obstacle. The blue region (CC_i) denotes the collision cone corresponding to the obstacle, a circle representing $\mathcal{B}(\boldsymbol{p}_i, r_i + r_R)$. Thus, v_r (the relative velocity of the robot to the obstacle) is infeasible, and another velocity (e.g., v) must be selected, such that $v \in \mathcal{V} \setminus CC_i$.

3 BACKGROUND

We now introduce the fundamentals of the VO paradigm and Monte Carlo planning for MDPs, which are the base of the methodology described in this paper.

3.1 Velocity Obstacles (VO)

In the classical VO setting, a robot R must reach a target G in an environment with N obstacles. Without loss of generality, we assume that the robot and the obstacles are spherical[3], with radii r_R and r_i, $i = 1, \ldots, N$, respectively. At a given time step $\bar{t}$, the robot is at (vector) location $\boldsymbol{p}_R$, while the obstacles have positions $\boldsymbol{p}_i$ and velocity (vector) $\boldsymbol{v}_i$ (in our setting, the velocity of the obstacles is unknown). Given the set $\mathcal{V}$ of admissible velocities for robot R, the VO paradigm is used to compute the set of collision-free velocities $\mathcal{V}_c \subseteq \mathcal{V}$. Specifically, for each i-th obstacle, we define a *collision cone* CC_i as:

$$CC_i = \{\, \boldsymbol{v} \in \mathcal{V} \mid \exists t > \bar{t} \text{ s.t. } \boldsymbol{p}_R(\bar{t}) + \boldsymbol{v}_r(t - \bar{t}) \cap \mathcal{B}(\boldsymbol{p}_i, r_R + r_i) \neq \emptyset \} \quad (1)$$

where $\mathcal{B}(\boldsymbol{p}_i, r_R + r_i)$ is the ball centered at $\boldsymbol{p}_i$ with radius $r_R + r_i$, and $\boldsymbol{v}_r = \boldsymbol{v} - \boldsymbol{v}_i$ is the relative velocity of the robot with respect to the obstacle. We then define $\mathcal{V}_c = \mathcal{V} \setminus \bigcup_{i=1}^{N} CC_i$, the latter being the union of cones for every obstacle. In Figure 1, we show the velocity space of the robot for a simple scenario with the collision cone for one obstacle.

3.2 Markov Decision Processes

A Markov Decision Process (MDP) [6, 43] is a tuple $M = \langle S, A, T, R, \gamma \rangle$, where S is a finite set of *states* (e.g., robot and obstacle positions in the VO setting), A is a finite set of *actions* - we represent each action with its index, i.e., $A = \{1, \ldots, |A|\}$ (e.g., linear velocity and movement direction in the VO setting), $T : S \times A \to \mathscr{P}(S)$ is a stochastic or deterministic *transition function* (e.g., the deterministic dynamics of the robot in the VO setting), where $\mathscr{P}(E)$ denotes the space of probability distributions over the finite set E, therefore $T(s, a, s')$ indicates the probability of reaching the state $s' \in S$ after executing $a \in A$ in $s \in S$, $R : S \times A \times S \to [-R_{max}, R_{max}]$ is a bounded stochastic *reward function* (e.g., a function that rewards the robot if it gets close to or reaches the goal avoiding the obstacles, in the VO setting), and $\gamma \in [0, 1)$ is a *discount factor*. The set of stochastic policies for M is $\Pi = \{\pi : S \to \mathscr{P}(A)\}$. In the VO setting used in this paper, a policy is a function that suggests the linear velocity and direction of the movement given the current position of the robot and the obstacles. Given an MDP M and a policy π we can compute state values $V_M^\pi(s), s \in S$, namely, the expected value acquired by π from s; and action values $Q_M(s, a), s \in S, a \in A$, namely, the expected value acquired by π when action a is performed from state s. To evaluate the performance of a policy π in an MDP M, called $\rho(\pi, M)$ in the following, we compute its expected return (i.e., its value) in the initial state s_0, namely, $\rho(\pi, M) = V_M^\pi(s_0)$. The goal of MDP solvers [45, 48] is to compute optimal policies, namely, policies having maximal values (i.e., expected return) in all their states.

[3]More complex shapes can be considered, e.g., square obstacles [40].

3.3 Monte Carlo Tree Search

MCTS [9, 13] is an online solver, namely, it computes the optimal policy only for the current state of the agent. In particular, given the current state of the agent, MCTS first generates, in a sample-efficient way, a Monte Carlo tree rooted in the current state of the agent. In this way, it estimates the Q-values (i.e., action values) for that state. Then, it uses these estimates to select the best action. A certain number $m \in \mathbb{N}$ of simulations are performed using, at each step, Upper Confidence Bound applied to Trees (UCT) [5, 29] (inside the tree) or a rollout policy (from a leaf to the end of the simulation) to select the action. The transition model (or an equivalent simulator) is used to perform the step from one state to the next. Simulations allow updating two node statistics, namely, the average discounted return $Q(s, a)$ obtained by selecting action a from state s, and the number of times $N(s, a)$ action a was selected from state s. UCT extends UCB1 [5] to sequential decisions and allows to balance exploration and exploitation in the simulation steps performed inside the tree, and to find the optimal action as m tends to infinity. Given the average return $\bar{Q}_{a,T_a(t)}$ of each action $a \in A$ after t simulations, where $T_a(t)$ is the number of times action a has been selected up to simulation t, UCT selects the action with the best upper confidence bound. In other words, the index of the action selected at the t-th visit of a node is $I_t = \text{argmax}_{a \in 1,\ldots,|A|}\ \bar{Q}_{a,T_a(t)} + 2C_p\sqrt{\frac{\ln(t-1)}{T_a(t-1)}}$, with appropriate constant $C_p > 0$. When all m simulations are performed, the action a with maximum average return (i.e., Q-value) $\bar{Q}_{a,T_a(m)}$ in the root is executed in the real environment.

4 METHODOLOGY

We consider a 2D motion planning scenario, where the robot has only access to the current position of obstacles, their maximum speed and radius, hence requiring no strict communication capabilities. It is crucial to note that no specific knowledge is available about obstacle behaviors. This replicates a realistic robotic setting where an agent can rely only on its sensors (e.g., LIDARs or cameras) to estimate the current state of the surrounding environment. We define the problem with N dynamic obstacles and a goal in position $\boldsymbol{p}_G$ as a MDP with state $S = \{\langle \boldsymbol{p}_R, \boldsymbol{v}_R, \mathcal{P}_o, \mathcal{R}_o, \mathcal{V}_{max,o} \rangle\}$ and $A = \mathcal{V}$ (i.e., admissible velocities for the robot), where $\mathcal{P}_o = \{\boldsymbol{p}_i\}_{i=1}^{N}$

and $\mathcal{R}_o = \{r_i\}_{i=1}^N$ denote the list of obstacles positions and radii, respectively; $\mathcal{V}_{max,o}$ is the list of maximum speeds of the obstacles. Notice that the state space is in general continuous, because the robot can reach all possible positions in the environment, and the action space is also continuous (we will discretize it later to use it in MCTS). The reward for a given tuple $\langle s, a, s' \rangle$ (state, action and next state, respectively) is modeled as:

$$R(s,a,s') = \begin{cases} R_h & \text{if at } s',\ ||\boldsymbol{p}_G - \boldsymbol{p}_R||_2 < r_R \\ -R_h & \text{if at } s',\ \mathcal{B}(\boldsymbol{p}_R, r_R) \text{ out of } W_{lim} \\ -R_h & \text{if at } s',\ \exists i \text{ s.t. } ||\boldsymbol{p}_i - \boldsymbol{p}_R||_2 < r_R + r_i \\ -\frac{||\boldsymbol{p}_R - \boldsymbol{p}_G||_2}{d_{max}} & \text{otherwise} \end{cases} \tag{2}$$

where R_h is a high reward value[4], $\mathcal{B}(\boldsymbol{p}_R, r_R)$ is the ball centered at $\boldsymbol{p}_R$ with radius r_R, W_{lim} denotes the limits of the workspace and d_{max} is the maximum distance to the goal in the given map. Equation (2), rewards goal reaching (first line), penalizes going out of workspace bounds (second line) or hitting obstacles (third line), and penalizes the normalized distance to the goal (last line). The transition map T is deterministic. The main idea of the proposed methodology is to introduce the VO constraint in the simulation process performed by MCTS to estimate action values. This can improve the efficiency of the simulation process, allowing only exploration of $\mathcal{V}_c \subseteq \mathcal{V}$ (i.e., safe collision-free velocities).

4.1 Action space discretization

We express each velocity $\boldsymbol{v} \in \mathcal{V}$ as a tuple $\boldsymbol{v} = \langle v, \alpha \rangle$, where v is the module of the velocity and α is the heading angle in radians. We assume that the physical constraints of the robot impose a maximum velocity module v_{max} and a maximum angular velocity ω_{max}. Thus, at each time step t_s, where the robot has heading angle α_0, the action space can be expressed as $A = \{\langle v, \alpha \rangle \mid 0 \le v \le v_{max}, \alpha_0 - \omega_{max} t_s \le \alpha \le \alpha_0 + \omega_{max} t_s\}$. We then obtain the action space for MCTS by discretizing v and α within their respective ranges[5] and considering all possible combinations of them. We also add actions in the form $\langle 0, \alpha \rangle$, to allow in-place rotation.

4.2 Integration of VO into MCTS

We introduce VO in two phases of MCTS, namely, Monte Carlo tree exploration, where UCT selects actions in simulation steps performed inside the Monte Carlo tree (in the following, this method will be called MCTS_VO_TREE) and rollout, where a random policy selects actions in simulation steps performed out of the Monte Carlo tree (this method is called MCTS_VO_ROLLOUT in the following). In both cases, we build collision cones as in Eq. (1), but considering only collisions happening within the duration of a time step t_s in the simulation [14], which is the discretization time step to compute the transition between subsequent states in the MDP. To reduce the computational complexity in finding the set of collision-free velocities $\mathcal{V}_c$, we consider the worst-case scenario where the module of the robot's velocity is v_{max}. In fact, if the robot cannot reach the obstacle at v_{max}, even lower velocities will be safe. Similarly, we assume that obstacles move at velocity with module $v_{max,i} \in \mathcal{V}_{max,o}$. In this way, the agent only needs position information about the obstacles (which are easier to estimate from sensors) instead of their velocity [17, 53]. In this conservative scenario, computing the set of safe (i.e., collision-free) velocities is equivalent to compute the set of safe heading angles $\mathcal{A}_c$ as indicated in Lines 8-9 of Algorithm 1. When assessing potential collisions, Algorithm 1 examines intersections between the collision cones formed by the agent and obstacles, represented by the intersection of balls ($\mathcal{B}(\boldsymbol{p}_R, r_1) \cap \mathcal{B}(\boldsymbol{p}_i, r_2)$) (Lines 5-7). When the intersection is non-empty, we compute lines from $\boldsymbol{p}_R$ to the intersections between the collision cones [6] (Line 8 and Figure 2a). Angles within these lines denote directions of potential collision at maximum speed. Consequently, outside angles are deemed safe for the agent to navigate at maximum speed (Line 14). In cases where no safe angles are available within the environment, the only viable option for the robot is to remain stationary (Line 16).

Algorithm 1 Compute $\mathcal{V}_c$

```
1:  function COMPUTE_VEL(r_R, p_R, R_o, P_o, v_max, α_0, V_max,o, t_s)
2:      % Set of feasible angles within kinematic limits
3:      A_c ← {α | α_0 − ω_max t_s ≤ α ≤ α_0 + ω_max t_s}
4:      for p_i ∈ P_o ∧ r_i ∈ R_o ∧ v_max,i ∈ V_max,o do
5:          r_1 ← v_max t_s
6:          r_2 ← r_i + r_R + v_max,i t_s
7:          if B(p_R, r_1) ∩ B(p_i, r_2) ≠ ∅ ∧ p_R ∉ B(p_i, r_2) then
8:              ⟨α_1, α_2⟩ ← tangent angles from p_R to B(p_i, r_2)
9:              A_c ← A_c \ [α_1, α_2]
10:         else if p_R ∈ B(p_i, r_2) then
11:             A_c ← ∅
12:             break
13:     if A_c ≠ ∅ then
14:         V_c ← [0, v_max]
15:     else
16:         V_c ← {0}
    return V_c = V_c × A_c
```

Algorithm 2 UCT expansion with VO constraints

Require: radius of robot r_R, list of radii of obstacles $\mathcal{R}_o$, position of robot $\boldsymbol{p}_R$, list of positions of obstacles $\mathcal{P}_o$, current heading angle α_0, maximum agent linear and angular velocity modules v_{max}, ω_{max}, maximum obstacle velocities $\mathcal{V}_{max,o}$, simulation time step t_s

```
1: function EXPAND(n = ⟨s, a⟩)
2:     V_c = V_c × A_c ←COMPUTE_VEL(r_R, p_R, R_o, P_o, v_max, α_0)
3:     Choose new a' ∈ V_c with UCT [10]
4:     s' ~ T(s, a')
5:     Add new child n' = ⟨s, a'⟩ to n
6:     return n'
```

4.2.1 VO in UCT. It concerns simulation steps taken inside the Monte Carlo tree, namely, the first steps performed near the robot's current position, to realize safe collision avoidance in the short

[4] $R_h = 100$ in our experiments.
[5] We discretize v into 5 equally spaced values and α into 11 equally spaced values
[6] In case of walls, we model them as segments and compute lines from $\boldsymbol{p}_R$ to their extremes.

Algorithm 3 Rollout

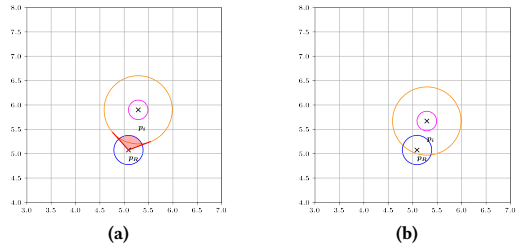
Require: radius of robot r_R, list of radii of obstacles $\mathcal{R}_o$, position of robot $\boldsymbol{p}_R$, list of positions of obstacles $\mathcal{P}_o$, current heading angle α_0, maximum agent linear and angular velocity modules v_{max}, ω_{max}, maximum obstacle velocities $\mathcal{V}_{max,o}$, simulation time step t_s, uniform selection threshold ϵ_0, heading direction to the goal α_G,

1: **function** ROLLOUT(state s)
2: **while** s is not terminal **do**
3: **if** Using VO **then**
4: $\mathcal{V}_c = V_c \times \mathcal{A}_c \leftarrow$ COMPUTE_VEL($r_R, \boldsymbol{p}_R, \mathcal{R}_o, \mathcal{P}_o, v_{max}, \alpha_0$)
5: **else**
6: $\mathcal{V}_c = V_c \times \mathcal{A}_c \leftarrow$ space of feasible actions
7: choose $\epsilon \in [0, 1]$ uniformly random
8: **if** $\epsilon \leq \epsilon_0$ **then**
9: choose $a \in \mathcal{V}_c$ uniformly random
10: **else**
11: choose $\alpha \in [\alpha_G - \delta, \alpha_G + \delta] \cap \mathcal{A}_c$ uniformly random
12: choose $v \in V_c$ uniformly random
13: $a \leftarrow [v, \alpha]$
14: $s' \sim T(s, a)$

term of the trajectory execution. Algorithm 2 shows the pseudo-code implementation of branch expansion in UCT based on VO[7]. Considering the maximum possible relative velocity between the agent and other robots/obstacles, at each step of UCT we compute the set $\mathcal{V}_c$ of velocity vectors that do not lead to collisions (Line 2 in Algorithm 2, using Algorithm 1), i.e., the set of vectors such that the resulting relative velocity $\boldsymbol{v}_r$ does not belong to any of the collision cones in Equation (1) within t_s. In this way, we prune away all unsafe actions, reducing the size of the action space and then making simulations more efficient. We remark that the UCT phase is where the agent selects the optimal action to execute in the real environment. Hence, VO in this phase is crucial for safe mobile robot navigation. In an ablation study (see next section), we found that introducing VO in the UCT phase (algorithm MCTS_VO_TREE in Section 5) yields the most promising results compared to other implementations. Therefore, we designate this configuration as our benchmark for comparison with the baselines.

4.2.2 VO in Rollout. The rollout phase of MCTS is known to be computationally demanding, hence it is important to design suitable heuristics to guide action selection [38, 44] and shield undesired or unsafe actions [37]. As a rollout policy (Algorithm 3), we use an heuristic to encourage the robot to move in the direction of the goal but also ensure convergence guarantees. Specifically, if the direction between the robot and the goal corresponds to angle α_G, we sample angles within $\mathcal{A}_c$ with uniform distribution in $[\alpha_G - \delta, \alpha_G + \delta]$, as in Line 9, being δ an empirical parameter ($\delta = 1$ rad in this paper). However, since the robot may get stuck due to obstacles on the way to the goal, we implement an ϵ-greedy strategy (Line 6) to follow this heuristic with probability $1 - \epsilon_0$. The rollout policy with the VO constraints (Line 4 of Algorithm 3), instead of sampling from all

[7]For the full UCT algorithm, please refer to [10].



(a) **(b)**

Figure 2: a) The robot ($\boldsymbol{p}_R$) is out of the extended ball $\mathcal{B}(\boldsymbol{p}_i, r_2)$ (yellow circle); b) The robot is inside the extended ball, but still not colliding with the physical ball of the obstacle $\mathcal{B}(\boldsymbol{p}_i, r_i)$ (pink circle). Blue circle: $\mathcal{B}(\boldsymbol{p}_R, r_1)$; red cone: CC_i delimited by tangent angles $[\alpha_1, \alpha_2]$ (Algorithm 1, Line 8).

the space, samples only safe angles within $\mathcal{A}_c$ by building collision cones and performing action pruning similarly to Algorithm 2.

4.3 Assumptions for safe collision avoidance

Introducing VO in MCTS allows to prune colliding actions (velocities), hence improving the planning efficiency and reducing the rate of collision with obstacles. In this paper, we assume minimal knowledge about the environment (i.e., only positions and maximum possible velocities of obstacles) for the best adherence to real-world robotic settings, showing the significant advantages of our methodology experimentally in the next section. However, the algorithm cannot *always guarantee* safe collision avoidance because collisions also depend on the behavior of dynamic obstacles. We now discuss in more detail our assumptions and necessary modifications to them in order to achieve formal guarantees.

4.3.1 Knowledge of $\mathcal{V}_{max,o}, \mathcal{P}_o$. Our methodology assumes that the positions and maximum velocities of obstacles are available to the agent at each step of MCTS computation. In general, obstacle positions can be estimated with standard robotic sensors, e.g., LIDARs, while estimating actual velocities is more challenging due to additional noise in the computation [50]. On the contrary, estimating the *maximum* possible velocities is easier, since they can be derived from available rough prior information (e.g., crowd models [27, 35] or other robots' specifications). Moreover, when computing safe velocities in Algorithm 1, we can enlarge $r_{1,2}$ (Lines 5-6) with safety bounds, in order to incorporate the level of confidence of sensors and uncertainties about $\mathcal{V}_{max,o}$. Note that, in case of large uncertainty, $\mathcal{A}_c = \emptyset$ in Algorithm 1. In this case, our algorithm is still able to take a collision-free action, which is remaining stationary (Line 16).

4.3.2 Unknown obstacle trajectories. Starting from a configuration where the center of the robot is outside the extended circumference $\boldsymbol{p}_R \notin \mathcal{B}(\boldsymbol{p}_i, r_2)$, see Figure 2a, we can guarantee collision avoidance one step ahead, using Algorithm 1. However, there may be situations where Algorithm 1 cannot compute tangent angles to the obstacle (Line 8), i.e., when $\boldsymbol{p}_R \in \mathcal{B}(\boldsymbol{p}_i, r_2)$[8] (see Figure 2b).

[8]This situation does not necessarily entail collision. Indeed, the actual collision occurs when the robot intersects with the physical radius of the obstacle, i.e., $\mathcal{B}(\boldsymbol{p}_R, r_R) \cap \mathcal{B}(\boldsymbol{p}_i, r_i)$.

This typically occurs when the obstacle moves too close towards the robot. Such situation cannot be taken into account by our algorithm, since we do not assume any prior knowledge about the intended trajectories of obstacles. Even in this condition, our algorithm guarantees that the robot does not take colliding actions, commanding null velocity (Lines 11 and 16 in Algorithm 1) but the obstacle could hit the robot. This situation should, however, not occur frequently in practical settings, e.g., in crowd navigation people tend to preserve a minimum distance to minimize the risk of collisions [27]. Hence, our approach works in standard settings. We will consider adversarial situations in which *malicious* obstacles *deliberately* try to collide with the robot in future works.

4.3.3 Simulation time. In Algorithm 1, safe velocities are computed assuming that the obstacles move at $\mathcal{V}_{max,o}$ during the simulation time step t_s. Hence, collision avoidance is only guaranteed if the planning time step (i.e., the time required by MCTS to compute the optimal action to be executed) is lower than t_s (i.e., the step time in MCTS, needed to compute the action space A in Section 4.1). In our experiments, we will show that VO action pruning significantly increases the efficiency of MCTS, thus realizing this requirement for safe collision avoidance.

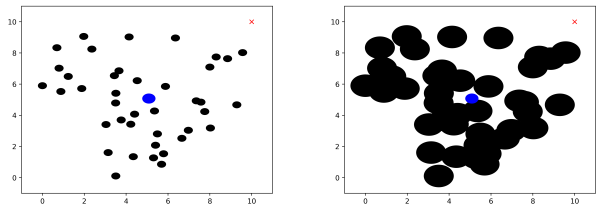
5 EXPERIMENTAL RESULTS

We evaluate each algorithm with a number of MCTS simulations ranging in $[10, 400]$[9] and investigate the performance as descibed in Section 5.3 in terms of *discounted return*, *collision rate*, and *computational time per step*. To account for the stochasticity of MCTS, we run 50 tests for each number of simulations. In the action space, we consider 60 actions, which combine 5 different velocity modules up to robot's v_{max} and 12 different heading angles among the feasible ones. The discount factor for MDP is chosen empirically as $\gamma = 0.7$. In the rollout phase, we empirically set $\epsilon_0 = 0.2$ in Algorithm 3. The maximum number of allowed steps in simulation is set to 100 (i.e tree depth + rollout steps). All experiments are run with Python 3.10 on a PC with Processor Intel(R) Core(TM) i5-13600KF CPU, 64GB Ram running Ubuntu 22.04.2 LTS. The code is available at https://github.com/Isla-lab/SafeMotionPlanningMCTS-VO.

5.1 Domain

We evaluated our methodology in the simulated map depicted in Figure 3a, consisting of a 10×10m workspace containing 40 dynamic obstacles. For dynamic obstacles modeling, we set all maximum velocities in $\mathcal{V}_{max,o}$ as $v_{max,o} = 0.2\,\mathrm{m/s}$. At each time step, dynamic obstacles move towards randomly predefined goals on the map, with additive noise[10]. Specifically, at each time step the velocity is sampled uniformly in $[-\frac{v_{max,o}}{2}, \frac{v_{max,o}}{2}]$; then the heading angle is the obstacle's goal direction, plus uniform noise in $[-0.05, 0.05]$. For robot modeling, we set realistic values $v_{max} = 0.3\,\mathrm{m/s}$ and $\omega_{max} = 1.9\,\mathrm{rad/s}$ from the datasheet of Turtlebot 4 by Clearpath Robotics, which is an established open-source mobile robotic platform for research and education [11]. We performed our experiments

[9]Above 400 simulations per step, we did not notice significant variations in the results.
[10]We also successfully tested our methodology against benchmark deterministic trajectories for obstacles, e.g., trefoils proposed in [51]. Due to page limits, the results are reported in the supplementary materials.
[11]https://clearpathrobotics.com/turtlebot-4/



(a) **(b)**

Figure 3: Snapshot of the actual environment with obstacle radius $r_i = 0.2\,\mathrm{m}$ and robot radius $r_R = 0.3\,\mathrm{m}$ (a); and radii r_1, r_2 as of Algorithm 1. Blue circle: agent. Black circles: dynamic obstacles. Red cross: goal.

across 50 distinct scenarios, randomly varying the initial positions of the obstacles, to thoroughly assess the performance of our motion planning strategy. In all scenarios, the robot has radius $r_R = 0.3\,\mathrm{m}$, while each i-th obstacle has radius $r_i = 0.2\,\mathrm{m}$. The chosen setup deliberately represents a cluttered environment, constituting a challenging planning context that requires the adoption of an efficient and safe look-ahead planning strategy. In particular, the considered setting prevents the use of a standard map discretization approach, as it would prohibitively increase the computational complexity. This is evidenced in Figure 3b, where the VO-enlarged radii of obstacles and robot are shown (Lines 5-6 in Algorithm 1).

The discrete time step of motion is chosen empirically as $t_s = 1\,\mathrm{s}$, but it can be adapted depending on the specific map configuration and task needs.

5.2 Algorithms and Baselines

We assess the effectiveness of various look-ahead planners, operating on the premise of lacking knowledge about obstacle trajectory and without the need for prior offline training. The algorithms evaluated in our experiments are listed in the following:

- *MCTS_VO_TREE*: it is the algorithm introducing the VO constraint inside MCTS only in the simulation steps performed inside the tree (see Section 4.2.1). This is the version of our approach showing the best tradeoff between performance and computational time (see Section 5.5);
- *MCTS_VO_ROLLOUT*: it is the algorithm introducing the VO constraint inside MCTS only in the simulation steps performed during the rollout phase (see Section 4.2.2);
- *MCTS_VO_2*: it is the algorithm introducing the VO constraint both inside the Monte Carlo tree and in the rollout phase;
- *MCTS*: standard implementation of MCTS (without VO). This is used as a baseline to show the improvement achieved introducing VO within MCTS;
- *Non-linear Model Predictive Control (NMPC)*: a state-of-the-art planning algorithm based on nonlinear model predictive control and an established methodology for optimal motion planning [51]. In particular, it is a look-ahead planner which solves a constrained optimization problem over an horizon τ,

where the reward components in Equation (2) are converted into cost components (i.e the sign is inverted).

- *Dynamic Window Approach (DWA)*: a standard methodology for efficient optimal planning in dynamic environments [36].
- *VO_PLANNER*: a reactive planner based on VO and informed random exploration of angles of velocities towards the goal, following Lines 3-10 of Algorithm 3. This baseline is based on VO, hence provides the same theoretical guarantees as our methodology. However, as shown in the following experiments, it does not perform look-ahead planning, thus resulting in poorer performance in our challenging map.

5.3 Performance measures

To evaluate the performance of the algorithms, we use the three measures defined in the following:

- *Discounted Return* (ρ): it is the discounted sum of all rewards obtained during the course of a trajectory, i.e., $\rho = \sum_{k=0}^{H} \gamma^{k-1} r_k$, where H is the total number of steps in the trajectory. It indicates the quality of the trajectory and the travel distance, since at each step the agent cumulates a small negative reward at each step. This measure must be maximized.
- *Collision rate* (η): it is the ratio between the number of episodes in which the agent collides and the total number of episodes performed, i.e., $\eta = \frac{\sum_{e=1}^{n\_exp} \mathbb{1}_{collides}}{n\_exp}$. This measure must be minimized, as it quantifies the reliability of the planner across different environment settings.
- *Planning time per step* (t_{plan}): it is the average computational time taken by the planner to compute a planning step. It is useful to evaluate the applicability of each planning algorithm to real-world robotic setting, where the time available for deciding the next action is limited. This measure must be minimized and kept below t_s (see Section 4.3.3).

5.4 Comparison with baselines

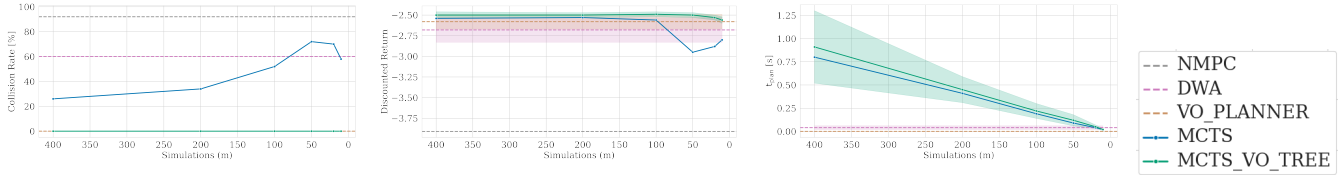
We compare the performance of MCTS_VO_TREE (i.e., the best version among the proposed approaches) with that of NMPC, DWA[12], VO_PLANNER and MCTS. Curves for NMPC (grey lines), DWA (pink lines) and VO_PLANNER (yellow lines) do not change with the number of simulations, since they are independent on that. Hyperparameter tuning was conducted for NMPC, in order to find the best time horizon $\tau \in [10, 70]$ to balance between computational efficiency and task performance, resulting in $\tau = 70$. Figure 4b[13] shows that MCTS_VO_TREE (green curve) outperforms all competitors in terms of discounted return (ρ), especially achieving a much narrower standard deviation, i.e., more robust behaviour across 50 random trials per m value. The performance remains superior also when very few simulations per step are made ($m < 20$) because VO focuses the search performed by MCTS on the collision-free action subspace avoiding useless simulations. Moreover, together with VO_PLANNER (which however achieves lower average return), MCTS_VO_TREE is the only algorithm ensuring safe collision avoidance for any number of simulations, under the assumptions in Section 4.3, as evidenced by Figure 4a. In particular, the low return achieved by NMPC is probably due to the highly cluttered environments and the high rate of collisions (> 80% from Figure 4a). Moreover, the MCTS collision-rate increases as the number of simulations decreases, evidencing the need for safe VO action pruning, especially in highly sub-optimal conditions (low value of m). It is interesting to analyze the success rate of the algorithms, namely, the percentage of experiments where the goal is reached within the maximum number of steps (100) without collisions (plots are reported in the supplementary material). Specifically, MCTS_VO_TREE reaches the goal in about 80% of the experiments, without significant variations as m decreases. VO_PLANNER is the second best-performing planner (70%), while other algorithms perform significantly worse, especially MCTS, which experiences a very low success rate of around 20% as m decreases. This proves the fundamental role of VO action pruning at efficiently guiding the agent towards the best and safest trajectories to the goal, but also the advantage of using a look-ahead planner instead of a reactive one as VO_PLANNER. Considering the computational performance (planning time t_{plan}), in Section 4.3.3 we mentioned that $t_{plan} < t_s = 1$ s in order to guarantee collision avoidance. From Figure 4c, MCTS_VO_TREE and MCTS achieve this when simulations $m < 300$, while DWA and VO_PLANNER are the most computationally efficient. However, as explained above, MCTS_VO_TREE significantly otuperforms the other algorithms in terms of collision rate, discounted return and success rate. In addition, from Figure 4b, MCTS_VO_TREE achieves high return even with very few simulations ($m < 20$), while MCTS performance significantly drop for $m < 100$. Hence, in practice MCTS_VO_TREE can achieve much better computational performance with respect to MCTS, working with fewer simulations (e.g., on average $t_{plan} = 0.2$ s with MCTS when $m = 100$, while $t_{plan} < 0.1$ s with MCTS_VO_TREE when $m < 50$). On the other hand, NMPC exhibits a substantially longer step time of 15.5 s, which makes it unusable with $t_s = 1$ s. To evaluate the quality of trajectories, we follow established literature [28] and measure the variation between consecutive speed commands (Linear velocity smoothness). MCTS_VO_TREE, VO_PLANNER and MCTS (having the highest success rate) achieve resp. 0.21 ± 0.02, 0.19 ± 0.03, and 0.19 ± 0.02, hence comparable within the std dev. Thanks to MDP formulation, the smoothness can also be easily improved by introducing a penalty for action variation in the reward.

5.5 Ablation study

To better understand the effects of the introduction of VO in different points of the MCTS algorithm, we compare the performance of four variants of the proposed algorithm, namely, MCTS_VO_TREE, MCTS_VO_ROLLOUT, MCTS_VO2, and MCTS. The results are depicted in Figure 5. Each line represents a different approach: blue for MCTS, orange for MCTS_VO2, red for MCTS_VO_ROLLOUT, and green for MCTS_VO_TREE. Figure 5b shows that MCTS_VO_TREE achieves the best discounted return on average, with the smallest standard deviation, even with very few simulations ($m < 20$), thus

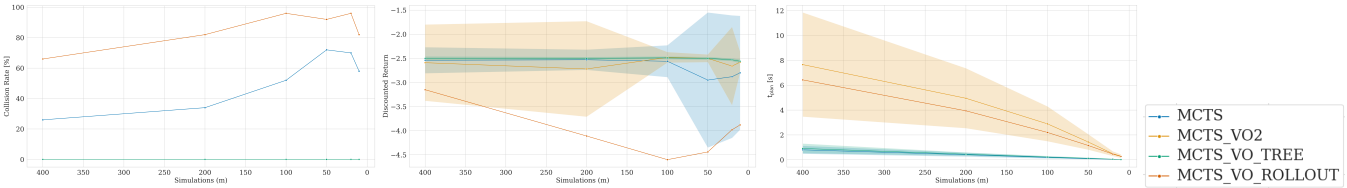
[12]We use the available Python implementations at https://github.com/atb033/multi_agent_path_planning (NMPC) [32] and https://github.com/AtsushiSakai/PythonRobotics/ (DWA)

[13]For NMPC, we omit the standard deviation since it is very large and affects readability. We include the plot with the standard deviation in the supplementary material.



(a) Collision Rate (η) (b) Discounted Return (ρ, mean ± std. dev.) (c) Planning Time per Step (t_{plan}, mean ± std. dev.) (d) Legend

Figure 4: Comparison between MCTS_VO_TREE, MCTS, NMPC, DWA and VO_PLANNER (m is the number of MCTS simulations). In Figure 4c we omitted NMPC since the value is out of scale compared to all other methods (average $t_{plan} \approx 10s$)



(a) Collision Rate (η) (b) Discounted Return (ρ) (c) Planning Time per Step (t_{plan}) (d) Legend

Figure 5: Results of the ablation study and comparison between MCTS_VO_TREE, MCTS, MCTS_VO2 and MCTS_VO_ROLLOUT.

confirming the results in Figure 4b. MCTS_VO2 has similar performance on average, but with higher standard deviation, hence being less stable. This is probably due to the lower success rate (< 70%, vs. 80% achieved by MCTS_VO_TREE). Indeed, VO in the rollout phase (Algorithm 3) limits the exploration of the action space, thus the agent more often chooses to remain stationary (null velocity) since no actions are available in the tree (Line 16 in Algorithm 1). MCTS_VO_ROLLOUT achieves the lowest discounted return, since it does not safely select velocities in UCT and the rollout exploration is limited by VO action pruning. Figure 5a shows that both MCTS_VO_TREE an MCTS_VO2 guarantee no collisions, even with $m = 10$ simulations. Indeed, in both algorithms actions are selected according the VO constraint in UCT. On the other hand, MCTS_VO_ROLLOUT and MCTS do not guarantee safe collision avoidance, and their performance downgrades with fewer simulations. In Figure 5c, the average planning time per step (t_{plan}) is illustrated. It is evident that integrating VO into the rollout phase (i.e., MCTS_VO_ROLLOUT and MCTS_VO2) significantly increases the computational time, primarily attributed to the computational cost incurred by the computation of $\mathcal{V}_c$ (Algorithm 1) at each simulation step. On the other hand, MCTS_VO_TREE only slightly increases the planning time step with respect to MCTS because it does not use VO in the computationally-expensive rollout phase. However, the computational time increase is absorbed by the advantages in terms of planning performance (discounted return and collision rate). Moreover, for all values of m, MCTS_VO_TREE always requires a computational time per step lower than t_s (i.e., < 1 s, which is a fundamental assumption to guarantee safe collision avoidance (Section 4.3.3). On the contrary, MCTS_VO2 only meets this requirement with a very low number of simulations ($m < 50$), where the discounted return is significantly less stable with higher standard deviation (Figure 5b). Finally, we remark that the computational efficiency is particularly important for practical deployment on real robots since the number of simulations affects the computational requirements on board of the physical system.

6 CONCLUSION AND FUTURE WORKS

We presented a novel algorithm for optimal online motion planning with collision avoidance in unknown dynamic and cluttered environments, combining the benefits of a look-ahead planner as Monte Carlo Tree Search (MCTS), with the safety guarantees about collision avoidance provided by Velocity Obstacles (VO). As evidenced by our ablation study, VO in the UCT phase of MCTS allows to significantly reduce the computational cost of the planner, restricting the action space to only collision-free actions and dramatically reducing the number of required online simulations. Moreover, we thoroughly discussed the assumptions required to guarantee safe collision avoidance, showing their feasibility in most practical robotic use cases. Notably, our algorithm does not require any prior knowledge about the trajectories of other obstacles, but only their positions at the planning time and maximum velocities. Validation in a 10×10m map with up to 40 randomly moving obstacles shows that our approach can compute high-quality trajectories with very few simulations per step in MCTS (less than 50), maintaining low variability in random scenarios (hence being more robust) and significantly outperforming several established competitors, including NMPC and DWA. In future works, we will investigate the extension of our methodology to continuous action spaces and partially observable MDPs , that present additional computational and modeling challenges but are of more practical interest in robotic domains.

ACKNOWLEDGMENTS

This work has been supported by PNRR MUR project PE0000013-FAIR

REFERENCES

[1] Guy Amir, Davide Corsi, Raz Yerushalmi, Luca Marzari, David Harel, Alessandro Farinelli, and Guy Katz. 2023. Verifying learning-based robotic navigation systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 607–627.

[2] Anton Andreychuk, Konstantin Yakovlev, Pavel Surynek, Dor Atzmon, and Roni Stern. 2022. Multi-agent pathfinding with continuous time. *Artificial Intelligence* 305 (2022), 103662.

[3] Luke Antonyshyn, Jefferson Silveira, Sidney Givigi, and Joshua Marshall. 2023. Multiple mobile robot task and motion planning: A survey. *Comput. Surveys* 55, 10 (2023), 1–35.

[4] Szilárd Aradi. 2020. Survey of deep reinforcement learning for motion planning of autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems* 23, 2 (2020), 740–759.

[5] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47, 2–3 (2002), 235–256.

[6] Richard Bellman. 1957. A Markovian decision process. *Journal of Mathematics and Mechanics* 6, 5 (1957), 679–684.

[7] Federico Bianchi, Lorenzo Bonanni, Alberto Castellini, and Alessandro Farinelli. 2022. Monte Carlo Tree Search Planning for continuous action and state space. In *Proceedings of the 9th Italian Workshop on Artificial Intelligence and Robotics (AIRO 2022) at AI*IA 2022 (CEUR Workshop Proceedings, Vol. 3417)*. CEUR-WS.org, 38–47.

[8] Federico Bianchi, Edoardo Zorzi, Alberto Castellini, Thiago D. Simão, Matthijs T. J. Spaan, and Alessandro Farinelli. 2024. Scalable Safe Policy Improvement for Factored Multi-Agent MDPs. In *Proceedings of the 41th International Conference on Machine Learning (ICML 2024) (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR, 3952–3973.

[9] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (2012), 1–43.

[10] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (2012), 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810

[11] Alberto Castellini, Federico Bianchi, Edoardo Zorzi, Thiago D. Simão, Alessandro Farinelli, and Matthijs T. J. Spaan. 2023. Scalable Safe Policy Improvement via Monte Carlo Tree Search. In *Proceedings of the 40th International Conference on Machine Learning (ICML 2023)*. PMLR, 3732–3756.

[12] Alberto Castellini, Enrico Marchesini, and Alessandro Farinelli. 2021. Partially Observable Monte Carlo Planning with state variable constraints for mobile robot navigation. *Engineering Applications of Artificial Intelligence* 104 (2021), 104382. https://doi.org/10.1016/j.engappai.2021.104382

[13] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. 2008. Monte-Carlo Tree Search: A New Framework for Game AI. In *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (Stanford, California) *(AIIDE'08)*. AAAI Press, 216–217.

[14] Daniel Claes, Daniel Hennes, Karl Tuyls, and Wim Meeussen. 2012. Collision avoidance under bounded localization uncertainty. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 1192–1198.

[15] Tuan Dam, Georgia Chalvatzaki, Jan Peters, and Joni Pajarinen. 2022. Monte-Carlo robot path planning. *IEEE Robotics and Automation Letters* 7, 4 (2022), 11213–11220.

[16] Lu Dong, Zichen He, Chunwei Song, and Changyin Sun. 2023. A review of mobile robot motion planning methods: from classical motion planning workflows to reinforcement learning-based architectures. *Journal of Systems Engineering and Electronics* 34, 2 (2023), 439–459.

[17] Lei Du, Floris Goerlandt, Osiris A Valdez Banda, Yamin Huang, Yuanqiao Wen, and Pentti Kujala. 2020. Improving stand-on ship's situational awareness by estimating the intention of the give-way ship. *Ocean Engineering* 201 (2020), 107110.

[18] Shang Erke, Dai Bin, Nie Yiming, Zhu Qi, Xiao Liang, and Zhao Dawei. 2020. An improved A-Star based path planning algorithm for autonomous land vehicles. *International Journal of Advanced Robotic Systems* 17, 5 (2020), 1729881420962263.

[19] Xiaojing Fan, Yinjing Guo, Hui Liu, Bowen Wei, and Wenhong Lyu. 2020. Improved artificial potential field method applied for AUV path planning. *Mathematical Problems in Engineering* 2020 (2020), 1–21.

[20] Paolo Fiorini and Zvi Shiller. 1998. Motion planning in dynamic environments using velocity obstacles. *The international journal of robotics research* 17, 7 (1998), 760–772.

[21] Philipp Foehn, Angel Romero, and Davide Scaramuzza. 2021. Time-optimal planning for quadrotor waypoint flight. *Science Robotics* 6, 56 (2021), eabh1221.

[22] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. 1997. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine* 4, 1 (1997), 23–33.

[23] Michele Ginesi, Daniele Meli, Andrea Calanca, Diego Dall'Alba, Nicola Sansonetto, and Paolo Fiorini. 2019. Dynamic movement primitives: Volumetric obstacle avoidance. In *2019 19th international conference on advanced robotics (ICAR)*. IEEE, 234–239.

[24] Michele Ginesi, Daniele Meli, Andrea Roberti, Nicola Sansonetto, and Paolo Fiorini. 2021. Dynamic movement primitives: Volumetric obstacle avoidance using dynamic potential functions. *Journal of Intelligent & Robotic Systems* 101 (2021), 1–20.

[25] Himanshu Gupta, Bradley Hayes, and Zachary Sunberg. 2022. Intention-Aware Navigation in Crowds with Extended-Space POMDP Planning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS, 562–570.

[26] Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. 2022. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems* 36, 1 (2022), 26.

[27] Ioannis Karamouzas, Brian Skinner, and Stephen J Guy. 2014. Universal power law governing pedestrian interactions. *Physical review letters* 113, 23 (2014), 238701.

[28] Jaroslaw Karwowski and Wojciech Szynkiewicz. 2023. Quantitative Metrics for Benchmarking Human-Aware Robot Navigation. *IEEE Access* 11 (2023), 79941–79953. https://doi.org/10.1109/ACCESS.2023.3299178

[29] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML 2006. 17th European Conference on Machine Learning (LNCS, Vol. 4212)*. Springer-Verlag, aa, 282–293.

[30] Jean-Claude Latombe. 2012. *Robot motion planning*. Vol. 124. Springer Science & Business Media.

[31] Kunming Li, Mao Shan, Karan Narula, Stewart Worrall, and Eduardo Nebot. 2020. Socially aware crowd navigation with multimodal pedestrian trajectory prediction for autonomous vehicles. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 1–8.

[32] Chang Liu, Seungho Lee, Scott Varnhagen, and H Eric Tseng. 2017. Path planning for autonomous vehicles using model predictive control. In *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 174–179.

[33] Li-sang Liu, Jia-feng Lin, Jin-xin Yao, Dong-wei He, Ji-shi Zheng, Jing Huang, and Peng Shi. 2021. Path planning for smart car based on Dijkstra algorithm and dynamic window approach. *Wireless Communications and Mobile Computing* 2021 (2021), 1–12.

[34] Anton Lukyanenko and Damoon Soudbakhsh. 2023. Probabilistic motion planning for non-Euclidean and multi-vehicle problems. *Robotics and Autonomous Systems* 168 (2023), 104487.

[35] Yuanfu Luo, Panpan Cai, Aniket Bera, David Hsu, Wee Sun Lee, and Dinesh Manocha. 2018. Porca: Modeling and planning for autonomous driving among many pedestrians. *IEEE Robotics and Automation Letters* 3, 4 (2018), 3418–3425.

[36] Sango Matsuzaki, Shinta Aonuma, and Yuji Hasegawa. 2021. Dynamic window approach with human imitating collision avoidance. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 8180–8186.

[37] Giulio Mazzi, Alberto Castellini, and Alessandro Farinelli. 2023. Risk-aware shielding of Partially Observable Monte Carlo Planning policies. *Artificial Intelligence* 324 (2023), 103987.

[38] Daniele Meli, Alberto Castellini, and Alessandro Farinelli. 2024. Learning Logic Specifications for Policy Guidance in POMDPs: an Inductive Logic Programming Approach. *Journal of Artificial Intelligence Research* 79 (2024), 725–776.

[39] MG Mohanan and Ambuja Salgoankar. 2018. A survey of robotic motion planning in dynamic environments. *Robotics and Autonomous Systems* 100 (2018), 171–185.

[40] Cristian Morasso, Daniele Meli, Yann Divet, Salvatore Sessa, and Alessandro Farinelli. 2024. Planning and Inverse Kinematics of Hyper-Redundant Manipulators with VO-FABRIK. In *European Robotics Forum*. Springer, 195–199.

[41] Zhenhua Pan, Chengxi Zhang, Yuanqing Xia, Hao Xiong, and Xiaodong Shao. 2021. An improved artificial potential field method for path planning and formation control of the multi-UAV systems. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 3 (2021), 1129–1133.

[42] Nicola Piccinelli, Federico Vesentini, and Riccardo Muradore. 2023. MPC-Based Motion Planning For Mobile Robots Using Velocity Obstacle Paradigm. In *2023 European Control Conference (ECC)*. IEEE, 1–6.

[43] Martin L. Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[44] Benjamin Riviere, Wolfgang Hönig, Matthew Anderson, and Soon-Jo Chung. 2021. Neural tree expansion for multi-robot planning in non-cooperative environments.

IEEE Robotics and Automation Letters 6, 4 (2021), 6868–6875.

[45] Stuart J. Russell and Peter Norvig. 2020. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, aa.

[46] Oren Salzman and Dan Halperin. 2016. Asymptotically near-optimal RRT for fast, high-quality motion planning. *IEEE Transactions on Robotics* 32, 3 (2016), 473–483.

[47] Wang Shaobo, Zhang Yingjun, and Li Lianbo. 2020. A collision avoidance decision-making system for autonomous ship based on modified velocity obstacle method. *Ocean Engineering* 215 (2020), 107910.

[48] Richard Sutton and Andrew Barto. 2018. *Reinforcement Learning, An Introduction* (2nd ed.). MIT Press, aa.

[49] Francesco Taioli, Francesco Giuliari, Yiming Wang, Riccardo Berra, Alberto Castellini, Alessio Del Bue, Alessandro Farinelli, Marco Cristani, and Francesco Setti. 2024. Unsupervised Active Visual Search with Monte Carlo Planning under Uncertain Detections. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 12 (2024), 11047–11058.

[50] Abhishek Thakur and P Rajalakshmi. 2024. L3D-OTVE: LiDAR-Based 3D Object Tracking and Velocity Estimation Using LiDAR Odometry. *IEEE Sensors Letters* (2024).

[51] Jesus Tordesillas and Jonathan P How. 2021. MADER: Trajectory planner in multiagent and dynamic environments. *IEEE Transactions on Robotics* 38, 1 (2021), 463–476.

[52] Charles W Warren. 1989. Global path planning using artificial potential fields. In *1989 IEEE International Conference on Robotics and Automation*. IEEE Computer Society, 316–317.

[53] Tianye Xu, Shuiqing Zhang, Zeyu Jiang, Zhongchang Liu, and Hui Cheng. 2020. Collision avoidance of high-speed obstacles for mobile robots via maximum-speed aware velocity obstacle method. *IEEE Access* 8 (2020), 138493–138507.

[54] Qingfeng Yao, Zeyu Zheng, Liang Qi, Haitao Yuan, Xiwang Guo, Ming Zhao, Zhi Liu, and Tianji Yang. 2020. Path planning method with improved artificial potential field—a reinforcement learning perspective. *IEEE access* 8 (2020), 135513–135523.

[55] Xinglong Zhang, Yan Jiang, Yang Lu, and Xin Xu. 2022. Receding-horizon reinforcement learning approach for kinodynamic motion planning of autonomous vehicles. *IEEE Transactions on Intelligent Vehicles* 7, 3 (2022), 556–568.