Monte Carlo planning for mobile robots in large action spaces with velocity obstacles

Lorenzo Bonanni*[1], Daniele Meli*[1], Alberto Castellini[1] and Alessandro Farinelli[1]

Abstract

Motion planning in dynamic environments is a challenging robotic task, requiring collision avoidance and real-time computation. State-of-the-art online methods as Velocity Obstacles (VO) guarantee safe local planning, while global planning methods based on reinforcement learning or graph discretization are either computationally inefficient or not provably collision-safe. In this paper, we combine Monte Carlo Tree Search (MCTS) with VO to prune unsafe actions (i.e., colliding velocities). In this way, we can plan with very few MCTS simulations even in very large action spaces (60 actions), achieving higher cumulative reward and lower computational time per step than pure MCTS with many simulations. Moreover, our methodology *guarantees* collision avoidance thanks to action pruning with VO, while pure MCTS does not. Results in this paper pave the way towards deployment of MCTS planning on real robots and multi-agent decentralized motion planning.

Keywords

Monte Carlo Planning, Velocity Obstacles, Markov Decision Processes, Robot Motion Planning

1. Introduction

Motion planning is a wide research area in robotics, especially mobile robotics. Autonomous navigation in dynamic, possibly uncertain, environments is a non-trivial task, which requires the computation of a safe trajectory towards a predefined goal, while avoiding obstacles and preserving computational efficiency for fast reaction to changes in the environment [1]. State-of-the-art approaches can be mainly categorized into *reactive, look-ahead* and *learning-based* planners. Reactive planners include the Velocity Obstacle (VO) paradigm [2, 3] and artificial potential fields [4, 5]. They compute the motion command at run time, evaluating the current configuration of the environment, e.g., obstacles, hence guaranteeing good performance even in dynamic scenarios. However, they present several limitations, e.g., they may get stuck in local minima in narrow or cluttered maps. On the contrary, look-ahead planners compute trajectories over a finite time horizon, hence evaluating a larger part of the environmental map and achieving higher convergence to the goal, even in complex maps. Exploring actions over a horizon, they can often generate *optimal plans*, according to user-defined criteria. State-of-the-art examples include graph-based planners as rapidly-exploring random trees [6] and Model Predictive Control (MPC) [7]. However, graph-based planners are typically computationally

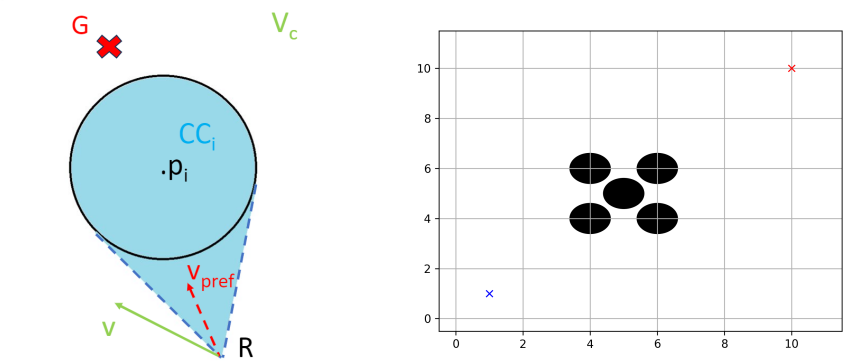
The 10th Italian Workshop on Artificial Intelligence and Robotics – AIRO 2023.

*Authors contributed equally.

*Corresponding author.

†These authors contributed equally.

lorenzo.bonanni@studenti.univr.it (L. Bonanni*); daniele.meli@univr.it (D. Meli*); alberto.castellini@univr.it (A. Castellini); alessandro.farinelli@univr.it (A. Farinelli)



(a) (b)

Figure 1: On the left, a sketch of the VO paradigm with one static obstacle. $\boldsymbol{v}_{pref} \in CC_i$, thus a new velocity $\boldsymbol{v} \in \mathcal{V}_c$ must be selected. **On the right**, the map of the environment used in this work, the agent has to move from the start point (blue cross) to the goal point (red cross), avoiding static obstacles (black circles).

inefficient in dynamic scenarios where replanning is needed. Moreover, MPC performs explicit optimization of a cost function, thus requiring an analytical definition which is not always easily available in complex scenarios [8]. Recently, learning-based planners adopting neural networks and reinforcement learning have been proposed [9], which are excellent solutions for optimal trajectory planning and are robust to uncertainty in the scenario. However, they require intense training resources (both data and time) and do not guarantee generalization to unseen scenarios. Finally, most optimal planners perform cost minimization, hence not guaranteeing safe navigation [10].

In this paper, we approximate the navigation problem as a Markov Decision Process (MDP), and solve it with Monte Carlo Tree Search (MCTS), an online optimal look-ahead planner which performs online simulations to compute the best trajectory. However, MCTS requires task-specific heuristics for computational efficiency [11, 12, 13, 14] and typically does not scale to large action spaces. We then combine it with the VO paradigm for collision avoidance. In particular, we *guarantee safe (i.e., obstacle-free) trajectory generation* by pruning colliding actions (i.e., velocities) in Monte Carlo tree and in the rollout phase. In this way, we are able to improve the performance of pure MCTS even with very few simulations, maintaining a large action space (60 actions) which allows a finer approximation of the environment. This is a preliminary but fundamental step towards deployment of Monte Carlo planning to real robots.

2. 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.

2.1. Velocity Obstacles (VO)

Consider a robot R which must reach a target G in an environment with N obstacles (an example with one obstacle is shown in Figure 1a). For simplicity, we assume that the robot and the obstacles are spherical, with radii r_R and $r_i, i = 1, \dots, N$. 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) $\boldsymbol{v}_i$. Given the set $\mathcal{V}$ of admissible velocities for robot R, this would move with velocity $\boldsymbol{v}_{pref} \in \mathcal{V}$ towards the goal, but this may cause collisions with obstacles. The VO paradigm is then 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:

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

where $\mathcal{B}(\boldsymbol{p}_i, r_R + r_i)$ is the ball centered at $\boldsymbol{p}_i$ with radius $r_R + r_i$. We then define $\mathcal{V}_c = \mathcal{V} \setminus \bigcup_{i=1}^{N} CC_i$, namely, the union of cones for every obstacle.

2.2. Markov Decision Processes

A Markov Decision Process (MDP) [15, 16] 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., $A = \{1, \dots, |A|\}$ (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 $s \in S$, $R : S \times A \to [-R_{max}, R_{max}]$ 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, i.e., $\rho(\pi, M)$, 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 [17, 18] is to compute optimal policies, namely, policies having maximal values (i.e., expected return) in all their states.

2.3. Monte Carlo Tree Search

MCTS [19, 20] is an online solver, namely, it computes the optimal policy only for the current state of the agent. This feature of MCTS allows it to scale to large state spaces, which are typical of real-world domains. Given the current state of the agent, MCTS first generates a Monte Carlo tree rooted in the state to estimate in a sample-efficient way 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 is performed using, at each step, Upper Confidence Bound applied to Trees [21, 22] (inside the tree) or a rollout policy (from a leaf to the end of the simulation) to select the

action, and the transition model (or an equivalent simulator) to perform the step from one state to the next. Simulations allow to update two node statistics, namely, the average discounted return $Q(s,a)$ obtained selecting action a, and the number of times $N(s,a)$ action a was selected from node (state) s. UCT extends UCB1 [21] 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 $\bar{X}_{a,T_a(t)}$ of each action $a \in A$ of a node, where $T_a(t)$ is the number of times action a has been selected up to simulation t from that node, 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 $I_t = \text{argmax}_{a \in 1,\dots,|A|} \; \bar{X}_{a,T_a(t)} + 2C_p\sqrt{\frac{\ln(t-1)}{T_a(t-1)}}$, with appropriate constant $C_p > 0$. When all m simulations are performed, the action a with maximum average return $\bar{X}_{a,T_a(t)}$ (i.e., Q-value) in the root is executed in the real environment.

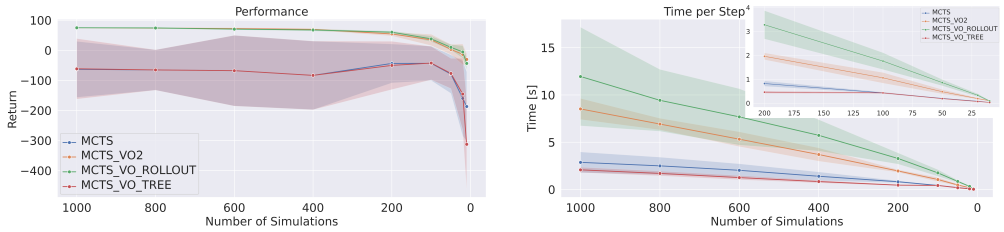
3. Methodology

We define the motion planning problem with N obstacles as a MDP with $S = \{\langle \boldsymbol{p}_R, \boldsymbol{v}_R, \boldsymbol{p}_i \; \forall i = 1,\dots,N\rangle\}$ and $A = \mathcal{V}$. Notice that the state space is continuous because the robot can reach all possible positions in the environment. For simplicity, we assume static obstacles ($\boldsymbol{v}_i = \boldsymbol{0}$) in this paper. 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 $\mathcal{V}_c \subseteq \mathcal{V}$.

In order to solve the MDP problem with MCTS, we first discretize the continuous action space V. Specifically, we express each $\boldsymbol{v} \in \mathcal{V}$ 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 minimum and maximum velocity module, respectively v_{min}, 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 discrete action space can be expressed as $A = \{\langle v, \alpha\rangle \mid v_{min} \leq v \leq v_{max}, \alpha_0 - \omega_{max}t_s \leq \alpha \leq \alpha_0 + \omega_{max}t_s\}$. We then obtain the action space for MCTS by discretizing v and α and considering all possible combinations of them. We also force actions in the form $\langle 0, \alpha\rangle$, in order to allow the robot to only rotate without moving.

We introduce VO in two phases of MCTS, namely, in the steps performed inside the Monte Carlo tree (where UCT is used to select the action) and in the steps performed outside the Monte Carlo tree (where the rollout policy is used to select the action). The first phase concerns simulation steps taken near the robot's current position, while the second phase concerns simulation steps performed also far from the robot's current position. In this way, not only we guarantee reactive collision avoidance within few steps of execution, but we also exploit VO when planning on a longer time horizon, improving the quality and safety of the overall trajectory to the goal. We build collision cones as in Eq. (1), but considering only collisions happening within t_s [23]. For computational simplicity, we consider the worst-case scenario where the module of the robot's velocity is v_{max}. We can then prune away all discrete angles in the action space which would lead to a collision with any obstacle, reducing the action space considered at each simulation step. If no angle is safe, we compute the cone for v_{min}. Since actions with null velocity are forced in A, at least one action will always be feasible.

As a rollout policy we use an heuristic to encourage the robot to move in the direction of



(a) (b)

Figure 2: On the left, mean and standard deviation of the discounted return for different number of simulations. **On the right**, mean and standard deviation of the computational time required by MCTS to compute an action for the next step.

the goal. Specifically, if the direction between the robot and the goal corresponds to angle α_G, we sample safe angles within $\mathcal{V}_c$ with uniform distribution in $[\alpha_G - 1, \alpha_G + 1]$. Since the robot may be stuck in maps with large obstacles on the way to the goal, we implement an ϵ-greedy algorithm, selecting a random angle in $\mathcal{V}_c$ with probability $\epsilon \in [0, 1]$, and following the heuristic with probability $1 - \epsilon$.

4. Results

Experimental setting. We test our planner in the map shown in Figure 1b. Our goal is to evaluate the advantage of introducing VO-constraints in MCTS. Videos with 2 and 4 moving agents (from which we can draw similar conclusions) are available at http://tinyurl.com/airo23bonanni

Algorithms evaluated. We compare the performance of standard MCTS (VANILLA) with that of MCTS with VO constraints. In particular, we make an ablation study of our methodology, considering MCTS with VO only in UCT (VO_TREE), MCTS with VO only in rollout (VO_ROLLOUT) and MCTS with VO both in UCT and rollout (VO2). We evaluate each algorithm with a number of simulations ranging in [10, 1000] to understand if VO-constraints improve simulation efficiency and, consequently, allow to reduce the number of simulations and the related simulation time. To account for the stochasticity of MCTS, we run 10 tests for each configuration.

Results and discussion. In Fig. 2a, we show the return achieved by the 4 algorithms considered in the analysis. The use of VO in rollout significantly improves the performance on average, in fact VO2 (orange line) and VO_rollout (green line) achieve similar results and both outperform MCTS (blue line) and MCTS_VO_TREE (red line). Also the standard deviation is lower, thus the behavior is more stable. Importantly, with less than 200 simulations there is a significant drop in the performance of MCTS and MCTS_VO_TREE, while with MCTS_VO2 and MCTS_VO_ROLLOUT the return remains high even with only 10 simulations. In Fig. 2b, we show the computational time per step required each algorithm. Computing collision cones in the rollout significantly increases the required time but the effect of this constraint on performance is also very strong, namely, VO2 and VO_ROLLOUT with only 10 simulations reach the same performance of VANILLA with 200 simulations. In this way, the time per step required by MCTS

with VO in the rollout is lower than that of VANILLA with equal performance (VO2 takes ≈ 0.1 s and has an average return of -30, while VANILLA requires ≈ 0.8 s and has an average return of -43). Moreover, we remark that VO2 *guarantees safe collision avoidance*, since VO prunes away unsafe actions, while such guarantee does not come with MCTS.

5. Conclusion

We presented a method for safe motion planning in dynamic environments, based on MCTS and the VO paradigm. MCTS computes a global trajectory towards the goal, limiting the risk of getting stuck in local minima with purely reactive planning algorithms. VO prunes colliding actions, guaranteeing safe obstacle avoidance, differently from common approaches based on reward shaping, and scaling to large action spaces (60 actions). Preliminary results in a simulated environment with static obstacles show that VO allows to significantly improve the performance of MCTS given the same time budget, or, equivalently, it allows to reach similar performance of standard MCTS with much less simulations (and related time per step). Our ablation study evidences the key role of VO in the rollout phase of MCTS. The very low number of required simulations allows to mitigate the complexity of computing collision cones at run time, achieving an average time per step of 0.1 s vs. 0.8 s required by pure MCTS. In the future, we will assess the performance of our algorithm in multi-agent settings with moving obstacles and more complex maps. Moreover, we plan to implement it on a real robot, further improving the computational time per step using C instead of Python.

Acknowledgement

This project has received funding from the Italian Ministry for University and Research, under the PON "Ricerca e Innovazione" 2014-2020" (grant agreement No. 40-G-14702-1).

References

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

[2] P. Fiorini, Z. Shiller, Motion planning in dynamic environments using velocity obstacles, The International Journal of Robotics Research 17 (1998) 760–772.

[3] F. Vesentini, N. Piccinelli, R. Muradore, Velocity obstacle-based trajectory planner for anthropomorphic arms, European Journal of Control (2023) 100901.

[4] C. W. Warren, Global path planning using artificial potential fields, in: 1989 IEEE International Conference on Robotics and Automation, IEEE Computer Society, 1989, pp. 316–317.

[5] M. Ginesi, D. Meli, A. Roberti, N. Sansonetto, P. Fiorini, Dynamic movement primitives: Volumetric obstacle avoidance using dynamic potential functions, Journal of Intelligent & Robotic Systems 101 (2021) 1–20.

[6] A. Lukyanenko, D. Soudbakhsh, Probabilistic motion planning for non-euclidean and multi-vehicle problems, Robotics and Autonomous Systems 168 (2023) 104487.

[7] C. E. Luis, M. Vukosavljev, A. P. Schoellig, Online trajectory generation with distributed model predictive control for multi-robot motion planning, IEEE Robotics and Automation Letters 5 (2020) 604–611.
[8] W. Edwards, G. Tang, G. Mamakoukas, T. Murphey, K. Hauser, Automatic tuning for data-driven model predictive control, in: 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2021, pp. 7379–7385.
[9] A. H. Qureshi, Y. Miao, A. Simeonov, M. C. Yip, Motion planning networks: Bridging the gap between learning-based and classical motion planners, IEEE Transactions on Robotics 37 (2020) 48–66.
[10] D. Kamran, T. D. Simão, Q. Yang, C. T. Ponnambalam, J. Fischer, M. T. Spaan, M. Lauer, A modern perspective on safe automated driving for different traffic dynamics using constrained reinforcement learning, in: 2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC), IEEE, 2022, pp. 4017–4023.
[11] D. Meli, A. Castellini, A. Farinelli, Learning logic specifications for policy guidance in pomdps: an inductive logic programming approach, Journal of Artificial Intelligence Research 79 (2024) 725–776.
[12] G. Mazzi, D. Meli, A. Castellini, A. Farinelli, Learning logic specifications for soft policy guidance in POMCP, in: Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS '23, IFAAMAS, 2023, p. 373–381.
[13] G. Mazzi, A. Castellini, A. Farinelli, Risk-aware shielding of Partially Observable Monte Carlo Planning policies, Artificial Intelligence 324 (2023) 103987.
[14] A. Castellini, E. Marchesini, A. Farinelli, Partially Observable Monte Carlo Planning with state variable constraints for mobile robot navigation, Engineering Applications of Artificial Intelligence 104 (2021) 104382.
[15] R. Bellman, A markovian decision process, Journal of Mathematics and Mechanics 6 (1957) 679–684.
[16] M. L. Puterman, Markov decision processes: discrete stochastic dynamic programming, John Wiley & Sons, 2014.
[17] S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Pearson, 2020.
[18] R. Sutton, A. Barto, Reinforcement Learning, An Introduction, 2nd ed., MIT Press, 2018.
[19] G. Chaslot, S. Bakkes, I. Szita, P. Spronck, Monte-Carlo Tree Search: A new framework for game ai, in: Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE'08, AAAI Press, 2008, p. 216–217.
[20] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, A survey of Monte Carlo Tree Search methods, IEEE Transactions on Computational Intelligence and AI in Games 4 (2012) 1–43.
[21] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Machine Learning 47 (2002) 235–256.
[22] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: Machine Learning: ECML 2006. 17th European Conference on Machine Learning, volume 4212 of *LNCS*, Springer-Verlag, 2006, pp. 282–293.
[23] D. Claes, D. Hennes, K. Tuyls, W. Meeussen, Collision avoidance under bounded localization uncertainty, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2012, pp. 1192–1198.